Dojumenation

Ouivg gUI progIraamning in windowc uuping application
Uoolnng C C++

Documentation for Windows Login System

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This document describes a Windows-based Login System created with the Windows API in C++. It is a straightforward authentication module, providing users with a secure and user-friendly interface to log into a Windows desktop application. The system includes essential functionalities like user credential input, login validation, error handling, and a registration prompt placeholder. This documentation is structured to guide developers, system administrators, and technical writers on the system's architecture, functionality, maintenance, and potential future improvements.

---

1. Introduction

- **Purpose**:

  This Login System module is designed to provide a basic user authentication interface for Windows desktop applications. It serves as the initial security checkpoint for users trying to access an application and provides feedback on login attempts.

- **Scope**:

  The scope of this documentation includes the design and functionality of the login interface, user input controls, error messages, and resource handling. It does not cover server-based authentication, database integration, or multi-factor authentication.

- **Audience**:

  Intended for software developers, system administrators, and technical writers interested in understanding and managing the login system within a Windows application.

---

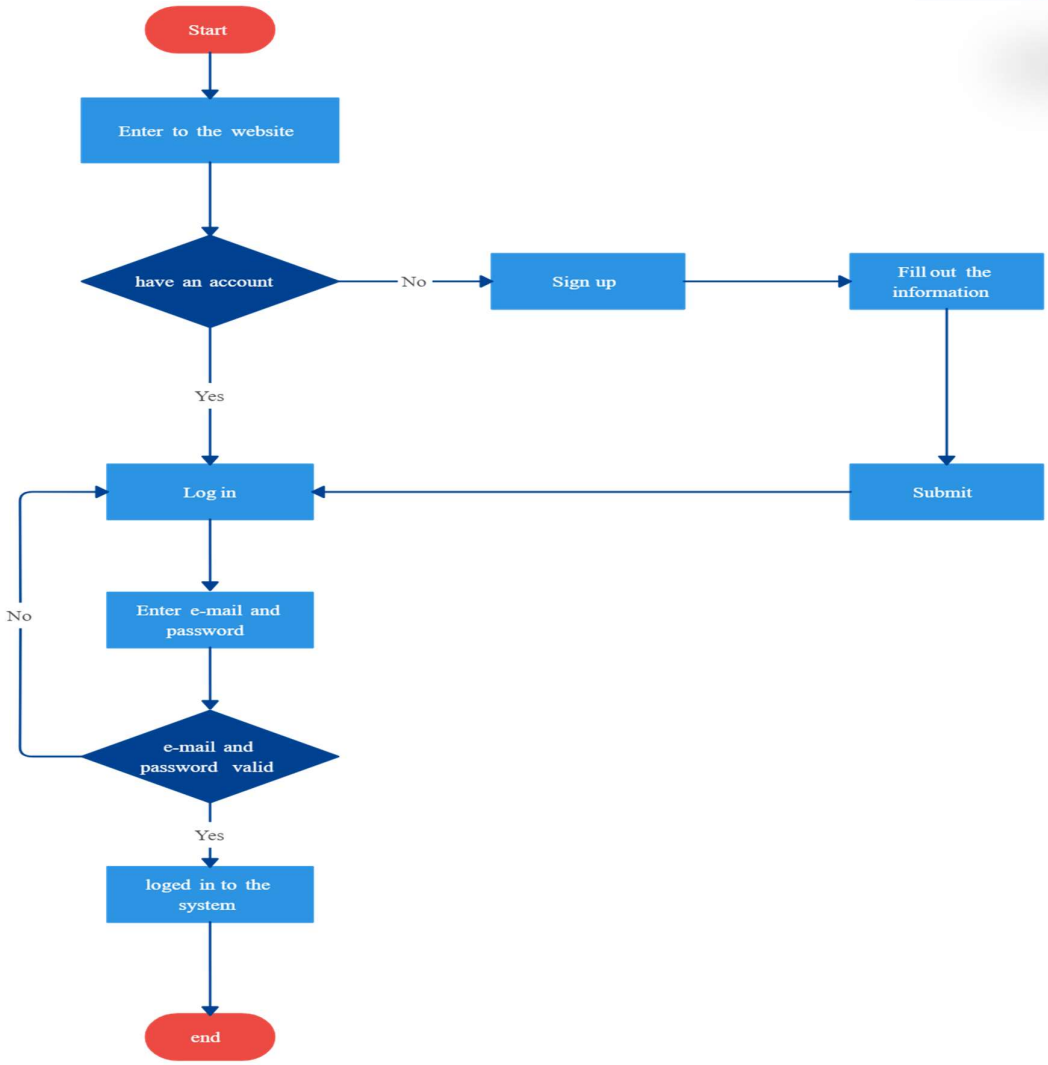
2. System Overview

- **High-LevelDescription**:

  The Login System is a client-side interface providing users with login functionality within a desktop application. It includes essential UI elements for user credential entry, validates these credentials, and offers feedback based on login success or failure. The system includes a fallback icon in case the login image cannot be loaded, enhancing robustness.

- **SystemArchitecture**:

  This system follows a monolithic architecture, operating as a single, self-contained module within a Windows application. It interacts with the Windows API to provide UI controls, manage user interactions, and handle events.



LogIn System Flowchart

Key Technologies:

- **Programming Language**: C++
- **Platform**: Microsoft Windows
- **Frameworks/Libraries**: Windows API
- **IDE**: DEVC++ IDE Windows API development.

---

3. Functional Specifications

- **User Roles and Permissions**:
  - **User**: Main role for application end-users who enter credentials to authenticate.
  - **Guest**: A view-only role, permitted to access the login interface but not other application areas.
- **Modules/Components**:
  - **Login Interface**:
    - *Description*: Displays username and password fields, with submit and register buttons.
    - *Interaction*: Users enter credentials and submit them for verification.
    - *Key Functionalities*: Credential validation, error and success messaging, and a registration feature placeholder.
  - **Footer**:
    - *Description*: Positioned at the bottom, displaying static copyright information.
    - *Interaction*: Displays non-interactive text.
    - *Key Functionalities*: Ensures brand visibility and compliance with copyright guidelines.
  - **Event Handlers**:
    - *Description*: Manages button click actions, including login attempts, resizing prevention, and cleanup.
    - *Key Functionalities*: Processes login requests, provides visual feedback, and maintains window integrity.
- **Use Cases**:
  - **User Login Attempt**: A user enters credentials, clicks Submit, and receives feedback based on validation results.

- **Registration Prompt**: On clicking the Register button, a placeholder message is displayed.
- **Centering the Window**: Upon launch, the window is centered on the screen for better usability.

- **Input/Output**:
  - **Input**:
    - Username: Text input.
    - Password: Protected text input (masked).
  - **Output**:
    - Success Message: "Login successful!"
    - Error Message: "Invalid username or password."
    - Registration Message: "Registration feature not implemented."

---

4. Non-Functional Specifications

- **Performance**:
  - The system should load within 2 seconds and handle interactions within 100 milliseconds.
  - Optimized memory use by using native Windows API elements.
- **Security**:
  - Masked password entry for basic privacy.
  - Placeholder for enhanced authentication methods (e.g., encryption or token-based validation).
- **Availability**:
  - Local desktop application ensures 100% availability without reliance on external services.
  - Can be backed up using standard system recovery tools.

- **Usability**:
  - Centered window and simple UI layout for ease of access.
  - Fixed window size prevents layout distortion or accidental resizing.
- **Reliability**:
  - Handles missing resources by falling back to default icons, preventing crashes.
  - Ensures clean exit by releasing resources like images and icons.

5. System Design

Architecture Design

This section provides an in-depth view of the system's architecture, including how the interface and processing components work together to create a seamless login experience.

- **Data Flow Diagrams (DFD):** The Data Flow Diagram (DFD) is used to illustrate how information moves through the login system. For example, when a user inputs their credentials, the data flows from the username and password fields into the login handling function, which then checks these inputs. Depending on the result, a success or error message is generated. This diagram helps explain the flow of data from the point of user input to the point where feedback (success or failure) is displayed.
- **Component Diagrams:** The component diagram illustrates how the system's elements connect and work together. Key components include:
  - **User Interface Elements** (text fields, buttons, labels): Capture and display information to the user.
  - **Event Handlers**: Process user actions like clicking "Submit" and handle validation for login attempts.

- **Resource Management**: Manages image and icon resources for the interface, providing visual elements for an enhanced user experience. The fallback mechanism for an image placeholder ensures that if an image cannot be loaded, a default icon is displayed instead, maintaining a cohesive look.

Database Design

This login system does not currently store user information in a database but is structured to support this functionality if needed in the future. Should the system integrate with a database for user data storage or log interaction, the design would include:

- **ER (Entity-Relationship) Diagrams**: These diagrams would represent tables like Users and AuditLog, showing the relationships between them. For instance, each login attempt could be logged to provide insight into user behavior or track invalid access attempts.
- **Tables and Schemas**: A preliminary schema design would define:
  - **Users Table**: Contains fields such as Username, Password, and Role (to differentiate user levels if needed).
  - **AuditLog Table**: Stores information on login attempts, with fields like EventID, Username, Timestamp, and EventType.

User Interface Design

The interface is designed with simplicity in mind. It includes:

- **Username and Password Fields**: Input areas for the user to enter their credentials.

- **Submit and Register Buttons**: Actions that allow users to attempt login or explore registration (currently a placeholder).
- **Feedback Messages**: On successful or unsuccessful login attempts, the user receives a message, guiding them to either retry or confirming successful login. This feedback ensures users know the system's response to their input.

API Design (if applicable)

If this login system were to integrate with an external authentication API, such as a third-party authentication service, this section would specify:

- **API Endpoints**: Paths to external APIs (e.g., /api/auth/login) for submitting credentials securely.
- **Data Formats**: JSON format or similar, for the structured exchange of information between the client and server.
- **Authentication Methods**: API keys or token-based authentication to securely manage user sessions.

---

6. System Implementation

Development Process

The login system was developed using a structured approach, resembling the Waterfall model. This process involved clear stages, including planning, interface prototyping, coding, and testing to ensure each step was thoroughly completed before moving to the next. The Waterfall model helped streamline development and allowed for early feedback at each stage to refine and optimize the system.

Configuration

Guidance on setting up and configuring the login system includes:

- **Installation Steps**: Detailed instructions on how to install the application for both developers and users on a Windows platform.
- **Environment Setup**: Required software configurations, such as the preferred IDE settings, the necessary Windows SDK, and the libraries required to compile and run the code.
- **Dependencies**: Includes WinAPI and other libraries essential for development, ensuring a smooth setup.

Code Structure

Understanding the code structure helps developers to navigate the source code and locate relevant files easily. Key components include:

- **main.cpp**: This file contains the main window procedure and handles the overall window creation and initialization.
- **Login.h and Login.cpp**: These files define and implement the login screen's interface and logic, handling input fields, button interactions, and success/error messages.
- **Resources/ Folder**: This directory houses resources like icons and images that enhance the UI, providing visual assets for the application.

Third-Party Integrations

Currently, the system leverages the Windows API to handle its graphical user interface, user input processing, and resource management. While it doesn't connect to any external systems or databases at the moment, future versions could potentially integrate with APIs for external authentication or cloud-based logging services.

7. Testing

Testing Strategy

To ensure a robust and reliable system, multiple testing methods were implemented:

- **Unit Testing**: Focuses on individual parts of the code, such as validating the functionality of each control (like buttons or text fields).
- **Integration Testing**: Examines how different parts of the system work together, especially the interactions between the UI and the underlying logic.
- **System Testing**: Tests the system as a whole, simulating typical user actions and responses.
- **User Acceptance Testing (UAT)**: Tests with actual users to confirm that the system meets user expectations and is intuitive.

Test Cases and Results

Testing involved a variety of scenarios:

- **Login Validation**: Verifying that correct credentials allow login while incorrect ones prompt an error.
- **Fallback Resource Check**: Ensuring that if the primary image cannot load, the system falls back to a default icon.
- **Error Message Display**: Testing to ensure informative error messages appear for failed login attempts.

Bug Tracking

A bug tracking system, such as a simple log or bug management software, was used to track any issues that arose. Examples include:

- **UI Misalignment**: Ensuring labels and input fields are correctly positioned.

- **Image Load Failure**: Ensuring the fallback mechanism works smoothly if an image file is missing.

---

8. Deployment & Maintenance

Deployment Instructions

The deployment process outlines how to set up the application on different environments, such as production or testing:

- **Installation Steps**: Includes steps to install executable files on Windows systems, making the application accessible to end-users.
- **Production Environment Setup**: Outlines additional configurations needed to ensure the system runs smoothly.
- **Development Environment**: Steps for setting up a testing environment where new features can be developed and tested safely.

System Requirements

To ensure smooth performance, the system has minimum requirements:

- **Operating System**: Windows 10 or later is recommended.
- **Hardware**: A system with 4GB RAM and a modern processor (Intel i3 or equivalent).
- **Software Dependencies**: Visual Studio or compatible IDE and the Windows SDK for development.

System Updates

This section outlines how to update and maintain the application:

- **Version Control**: Tracking updates using a version control system like Git helps manage changes and ensures safe updates.
- **Update Procedures**: Guidelines for applying new patches or updates while minimizing disruption.
- **Rollback Strategy**: In case of errors after an update, instructions to roll back to the previous stable version.

Monitoring

Monitoring tools, such as Windows Event Viewer, can be used to track the system's health:

- **System Logs**: Captures details about login attempts and errors.
- **Resource Usage**: Tracks memory and CPU usage, helping detect potential performance issues.

Maintenance

Regular maintenance ensures long-term stability:

- **Routine Backups**: Important for preserving configurations and logs.
- **Health Checks**: Periodic reviews of system performance to identify any emerging issues.
- **Data Cleanup**: Routine deletion of temporary files and unnecessary logs to maintain efficiency.

9. Troubleshooting & FAQs

Common Issues

Here are some common issues users may encounter, along with steps to resolve them:

1. **The login window won't open or crashes on startup.**
    - **Solution**: Ensure that all required files are present in the application directory. Reinstall the application to ensure there are no missing or corrupted files. If the problem persists, verify that your system meets the minimum requirements and that you have the necessary permissions to run the application.
2. **Error message: "Invalid credentials."**
    - **Solution**: Check that the username and password are entered correctly. Make sure Caps Lock is off, and no extra spaces are included. If you're certain your credentials are correct, contact system administration to verify your account status.
3. **The image fails to load, and no icon is displayed.**
    - **Solution**: The system should display a default icon if the primary image fails. If neither image appears, verify that the resource files are correctly located in the specified directory. Updating the application or reinstalling the resources might fix this issue.
4. **The window resizes unexpectedly despite fixed dimensions.**
    - **Solution**: If the window appears resized, this could be due to system settings or other applications affecting it. Ensure your screen resolution and scaling are set correctly. Restarting the application or adjusting display settings may help.
5. **Unresponsive buttons or fields.**
    - **Solution**: This might occur if the application lost focus or if there was a temporary loading issue. Try clicking outside the window and then back inside, or restart the application.

FAQs

1. **Can I resize the login window?**
    - No, the login window is set to a fixed size to maintain the interface design and ensure readability and usability across different systems.
2. **What happens if I enter incorrect login information?**
    - The system will display an error message, prompting you to re-enter your username and password. It does not lock the account after repeated attempts, but an account lock feature may be implemented in future versions.
3. **How do I retrieve or reset my password?**
    - Currently, this system does not support password recovery directly. Contact system administration for assistance in resetting your credentials.
4. **Can I customize the login screen?**
    - The login screen isn't customizable in this version, but future updates may allow for theme adjustments or other personalizations.
5. **How do I update the application?**
    - For most updates, replace the existing executable file with the new version provided. Future versions may introduce an automated update mechanism.

Logging and Debugging

To assist with logging and debugging:

- **Accessing Logs**: If logging is implemented, access log files from the designated folder, typically named logs in the application directory. These files provide information on recent login attempts, errors, and other system events.

- **Error Messages**: The application displays error messages in case of login issues or missing resources. Reviewing these messages can often give insight into potential problems.
- **Debugging Tools**: For developers, using Visual Studio's debugging tools or other compatible IDEs can assist in stepping through code, especially when investigating issues with UI behavior or event handling.

---

10. Glossary & References

Glossary

- **API**: Application Programming Interface; a set of functions that allows different software components to communicate.
- **Data Flow Diagram (DFD)**: A diagram that illustrates the flow of data within a system.
- **ER Diagram**: Entity-Relationship Diagram; represents the relationships between database tables.
- **UI**: User Interface; the part of the application that users interact with directly.
- **Event Handler**: A procedure in programming that processes events, such as clicks, keystrokes, or input submission.
- **WinAPI**: Windows Application Programming Interface, a core set of functions provided by Microsoft for creating Windows-based applications.
- **Fallback Mechanism**: A secondary option that the system uses if the primary resource (e.g., an image) is unavailable.

References:

BOOKS:

1. Kusswurm, D. (2024). Practical C++ STL Programming: Real-World Applications with C++20 and C++23. Springer.
2. Thomas, G. M. (2024). The Complete C++ Programming Guide: Master C++ Syntax, Data Structures, OOP, Templates, STL, Multithreading, and Design Patterns. Amazon Kindle Edition.
3. Soulié, J. (2007). The C++ Language Tutorial. Retrieved from https://cplusplus.com/files/tutorial.pdf

OTHERS:

1) Gibson, G. Essential C++ Programming for GUI Development with Qt: The Qt C++ Programmer's Handbook: Fundamental Skills for GUI Development, Thread Management and More. Amazon Kindle Edition.
2) Microsoft. Build desktop Windows apps using the Win32 API - Win32 apps. Retrieved from https://learn.microsoft.com/en-us/windows/win32/
3) Microsoft. Visual Studio C/C++ IDE and Compiler for Windows. Retrieved from https://visualstudio.microsoft.com/vs/features/cplusplus/
4) Programiz. Learn C++ Programming. Retrieved from https://www.programiz.com/cpp-programming
5) Udemy. Robust Qt & C++ GUI Programming 2D Graphics App Tutorial. Retrieved from https://www.classcentral.com/course/udemy-qt5-gui-cpp-programming-tutorial-2d-graphics-26786
6) Qt Project. Qt (software) - Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Qt_%28software%29

Appendix A

Login.cpp Source Code

```
// Login.cpp
#include <windows.h>
#include <string.h>
#include "Login.h"  // Include the header file

// Global variables for controls
HBITMAP hImage = NULL;
HICON hIcon = NULL;
HWND hUsernameLabel, hPasswordLabel;
HWND hUsernameField, hPasswordField;
HWND hSubmitButton, hRegisterButton;
HWND hFooterLabel;  // Footer label variable

// Function to create login form controls
void CreateLoginControls(HWND hwnd) {
    // First, we try to load an image to show on the left side of the login screen.
    // If that fails (maybe the file is missing or there's an issue), we fallback to a default
icon instead.
    hImage = (HBITMAP)LoadImage(NULL, "download.bmp", IMAGE_BITMAP, 0, 0,
LR_LOADFROMFILE);
    if (!hImage) {
        // If the image doesn't load, we load a default system icon as a backup.
        hIcon = LoadIcon(NULL, IDI_INFORMATION);
    }

    // Creating the Username label, which simply tells the user to enter their username.
    hUsernameLabel = CreateWindow("STATIC", "Username:", WS_VISIBLE |
WS_CHILD | SS_LEFT,
                                  310, 50, 80, 20, hwnd, NULL, NULL, NULL);

    // A text field where the user will type in their username.
    hUsernameField = CreateWindow("EDIT", "", WS_VISIBLE | WS_CHILD |
WS_BORDER,
                                  400, 50, 150, 20, hwnd, NULL, NULL, NULL);

    // The Password label, just like the username label, asks for the password.
    hPasswordLabel = CreateWindow("STATIC", "Password:", WS_VISIBLE |
WS_CHILD | SS_LEFT,
                                  310, 90, 80, 20, hwnd, NULL, NULL, NULL);

    // This is the password field, where the text is masked (hidden) as the user types.
    hPasswordField = CreateWindow("EDIT", "", WS_VISIBLE | WS_CHILD |
WS_BORDER | ES_PASSWORD,
                                  400, 90, 150, 20, hwnd, NULL, NULL, NULL);

    // The Submit button, which the user clicks to attempt logging in.
    hSubmitButton = CreateWindow("BUTTON", "Submit", WS_VISIBLE | WS_CHILD
| BS_DEFPUSHBUTTON,
                                  310, 140, 100, 30, hwnd, (HMENU)1, NULL, NULL);
```

```
    // The Register button, which currently just lets the user know that registration isn't
implemented.
    hRegisterButton = CreateWindow("BUTTON", "Register", WS_VISIBLE |
WS_CHILD | BS_PUSHBUTTON,
                                   450, 140, 100, 30, hwnd, (HMENU)2, NULL, NULL);

    // Creating the footer at the bottom of the window, which will display a copyright
message.
    CreateFooter(hwnd);
}

// Function to create the footer
void CreateFooter(HWND hwnd) {
    // The footer shows copyright information and stays fixed at the bottom of the
window.
    hFooterLabel = CreateWindow("STATIC", "© 2024 My App - All Rights Reserved",
WS_VISIBLE | WS_CHILD | SS_CENTER,
                                0, 300, 600, 20, hwnd, NULL, NULL, NULL);  // It spans the
entire width of the window
}

// Function to handle the login logic
void HandleLogin(HWND hwnd, WPARAM wParam) {
    switch (LOWORD(wParam)) {
        case 1: { // Submit Button
            // Here, we retrieve the username and password that the user has typed into
the fields.
            // The validation is simple: if the username is "user" and the password is
"pass", we log them in.
            char username[30], password[30];
            GetWindowText(hUsernameField, username, 30);
            GetWindowText(hPasswordField, password, 30);

            // If the login details are correct, we show a success message.
            // If not, we inform the user that the username or password is invalid.
            if (strcmp(username, "user") == 0 && strcmp(password, "pass") == 0) {
                MessageBox(hwnd, "Login successful!", "Success", MB_OK |
MB_ICONINFORMATION);
            } else {
                MessageBox(hwnd, "Invalid username or password.", "Error", MB_OK |
MB_ICONERROR);
            }
            break;
        }
        case 2: // Register Button
            // Since the registration feature isn't implemented, we show an informational
message.
            MessageBox(hwnd, "Registration feature not implemented.", "Info", MB_OK |
MB_ICONINFORMATION);
            break;
    }
}
// Function to handle painting the controls
void HandlePaint(HWND hwnd) {
```

```
    // This function is called when the window needs to be redrawn (for example, when
it's resized or uncovered).
    // We begin the painting process and get the device context (where we can draw).
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hwnd, &ps);

    // If the image was successfully loaded, we draw it on the left side of the window.
    if (hImage) {
        // We create a temporary drawing area (memory device context) to handle the
image.
        HDC hMemDC = CreateCompatibleDC(hdc);
        SelectObject(hMemDC, hImage);
        // Now, we copy the image onto the window's drawing area.
        BitBlt(hdc, 20, 50, 250, 200, hMemDC, 0, 0, SRCCOPY);
        // Clean up the temporary drawing area after we're done.
        DeleteDC(hMemDC);
    } else if (hIcon) {
        // If no image was loaded, we display the default icon instead.
        DrawIconEx(hdc, 20, 50, hIcon, 250, 200, 0, NULL, DI_NORMAL);
    }

    // Finally, we end the painting process to finalize everything.
    EndPaint(hwnd, &ps);
}

// Function to clean up when the window is destroyed
void Cleanup() {
    // When the window is being closed, we want to make sure that any resources
we've loaded (like images or icons) are properly cleaned up.
    // This helps prevent memory leaks.
    if (hImage) {
        DeleteObject(hImage);  // Free up the memory used by the image.
    }
    if (hIcon) {
        DestroyIcon(hIcon);  // Clean up the icon if we used it.
    }
}
```

Explanations:

1. Loading the Image or Icon: We first try to load an image to display on the left side of the screen. If that image isn't available, we fall back to showing a default icon instead.
2. Creating Controls: We set up the elements of the login form, including labels, text fields, and buttons. Each control has a specific purpose, like asking for the username or password or triggering the login action.
3. Login Logic: When the user clicks the "Submit" button, we check the entered username and password against hardcoded values. If they match, the user gets a success message; otherwise, an error message pops up.
4. Handling the Painting Process: When the window needs to refresh (for example, if it gets uncovered after being hidden), we redraw the elements like the image or icon. We ensure that the display is updated with what the user expects to see.
5. Cleanup: When the window is closing, we make sure to clean up any resources, such as the image or icon, that we used, so that we don't waste memory.

Appendix B

Login.h Source Code(Header File)

```
// Login.h

#ifndef LOGIN_H
#define LOGIN_H

#include <windows.h>

// This function will create all the necessary controls (like buttons, text fields, labels)
// for the login form and position them on the window.
void CreateLoginControls(HWND hwnd);

// This function is called when the user clicks the "Submit" button. It checks the
// entered username and password to see if they match predefined values,
// and handles the login process accordingly.
void HandleLogin(HWND hwnd, WPARAM wParam);

// This function is used to handle the drawing of the window's content,
// such as the image or icon, whenever the window needs to be redrawn.
void HandlePaint(HWND hwnd);

// This function cleans up any resources used during the window's lifetime
// (like images and icons) when the window is closed.
void Cleanup();

// This function creates and updates the footer at the bottom of the window,
// which displays the copyright message.
void CreateFooter(HWND hwnd);

#endif // LOGIN_H
```

Explanations:

1. **Creating the Login Controls**: This function sets up all the elements you'll see on the login screen: text fields for the username and password, labels, and buttons. It also places them in the window where they belong.
2. **Handling the Login**: When the user submits their login information, this function checks whether the entered username and password are correct. It helps decide whether to show a success or error message.
3. **Painting the Window**: This function is responsible for drawing the contents of the window, such as an image or icon, when the window is first shown or needs to be refreshed (e.g., after being resized).
4. **Cleanup**: Before the window closes, this function takes care of releasing any resources we've used (like the image or icon) to prevent memory leaks.
5. **Creating the Footer**: This function adds a footer to the window with a copyright message. It stays fixed at the bottom of the window.

Appendix C

Main.cpp Source Code

```
#include <windows.h>
#include "Login.h"  // Include the header file for login functionality

LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT msg, WPARAM
wParam, LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow) {
    const char className[] = "LoginWindowClass";

    // Setting up the window class with necessary properties.
    // This is essentially telling Windows how the window will behave.
    WNDCLASS wc = {};
    wc.lpfnWndProc = WindowProcedure;  // Link to the function that will handle the
messages
    wc.hInstance = hInstance;
    wc.lpszClassName = className;
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);  // Background color for
the window

    // Register the window class. If it fails, show an error message and exit.
    if (!RegisterClass(&wc)) {
        MessageBox(NULL, "Window Registration Failed!", "Error",
MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    // Define the window size as 600x350 so the login form fits comfortably.
    int windowWidth = 600;
    int windowHeight = 350;

    // Create the window, set it to be fixed-size (not resizable or maximizable).
    HWND hwnd = CreateWindowEx(
        0, className, "Login Panel", WS_OVERLAPPEDWINDOW &
~WS_MAXIMIZEBOX & ~WS_THICKFRAME,
        CW_USEDEFAULT, CW_USEDEFAULT, windowWidth, windowHeight,
        NULL, NULL, hInstance, NULL
    );

    // If the window creation fails, show an error and exit.
    if (!hwnd) {
        MessageBox(NULL, "Window Creation Failed!", "Error",
MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    // Get the dimensions of the screen (work area without taskbar)
    RECT rect;
    SystemParametersInfo(SPI_GETWORKAREA, 0, &rect, 0);
```

```
    // Calculate the coordinates to center the window on the screen
    int screenWidth = rect.right;
    int screenHeight = rect.bottom;
    int posX = (screenWidth - windowWidth) / 2;
    int posY = (screenHeight - windowHeight) / 2;

    // Set the window position so that it appears centered on the screen
    SetWindowPos(hwnd, HWND_TOP, posX, posY, windowWidth, windowHeight,
SWP_NOZORDER);

    // Set the window style to prevent resizing after creation
    SetWindowLong(hwnd, GWL_STYLE, GetWindowLong(hwnd, GWL_STYLE) &
~WS_THICKFRAME & ~WS_MAXIMIZEBOX);

    // Show the window and update its content
    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    // Enter the message loop to keep the application running
    MSG msg = {};
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);  // Convert the message into something that can be
processed
        DispatchMessage(&msg);   // Send the message to the window procedure
    }

    // Return the exit code of the application when it ends
    return msg.wParam;
}

// Window procedure is where all messages for the window are handled.
// It processes messages like clicks, resizing, and other events.
LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT msg, WPARAM
wParam, LPARAM lParam) {
    switch (msg) {
        case WM_CREATE:
            // When the window is created, call the function to add the login controls.
            CreateLoginControls(hwnd);
            break;

        case WM_COMMAND:
            // This handles user interactions with the login form, like button clicks.
            HandleLogin(hwnd, wParam);
            break;

        case WM_PAINT:
            // When the window needs to be redrawn, handle the painting (e.g., for
images).
            HandlePaint(hwnd);
            break;

        case WM_DESTROY:
            // Clean up resources when the window is closed, then post a quit message.
            Cleanup();
            PostQuitMessage(0);
```

```
            break;

        case WM_SIZE:
            // We block the window from resizing by intercepting the resize message.
            return 0;

        default:
            // If it's a message we're not handling, pass it to the default handler.
            return DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}
```

Explanations:

1. **WinMain Function**:
    - **Window Class Setup**: This part defines the properties of the window, like how it will look and how it will behave. For example, we associate the window class with a specific callback function that will handle events (like button clicks).
    - **Window Creation**: Here, the actual window is created with specific dimensions and fixed size (no resizing). If something goes wrong during this step, we show an error and exit.
    - **Centering the Window**: After creating the window, we calculate the screen's width and height and position the window right in the middle, so it looks good when it opens.
    - **Message Loop**: This is where the program waits for events, like button clicks or window resizing. It processes these events through the WindowProcedure function. This loop runs until the window is closed.
2. **WindowProcedure Callback**:
    - **Handling Different Messages**: This function is called every time the window needs to process an event (message). For example, when the window is created, it calls the CreateLoginControls function to set up the login form. If a button is clicked, it calls the HandleLogin function to check the login details.
    - **WM_CREATE**: When the window is created, we add the login controls like text fields and buttons.
    - **WM_COMMAND**: This handles events triggered by user interaction (like clicking the "Submit" button).
    - **WM_PAINT**: Whenever the window needs to redraw itself (for example, when it is first shown), this message ensures that the content is painted on the screen (like the image or icon).
    - **WM_DESTROY**: When the window is closing, we clean up any resources used by the window (like images) and then tell the application to exit.
    - **WM_SIZE**: This intercepts the window resizing action, ensuring that the window size remains fixed.
    - **Default Message Handling**: Any messages that aren't specifically handled are passed on to the default window procedure to ensure standard behavior.

Appendix D

Output

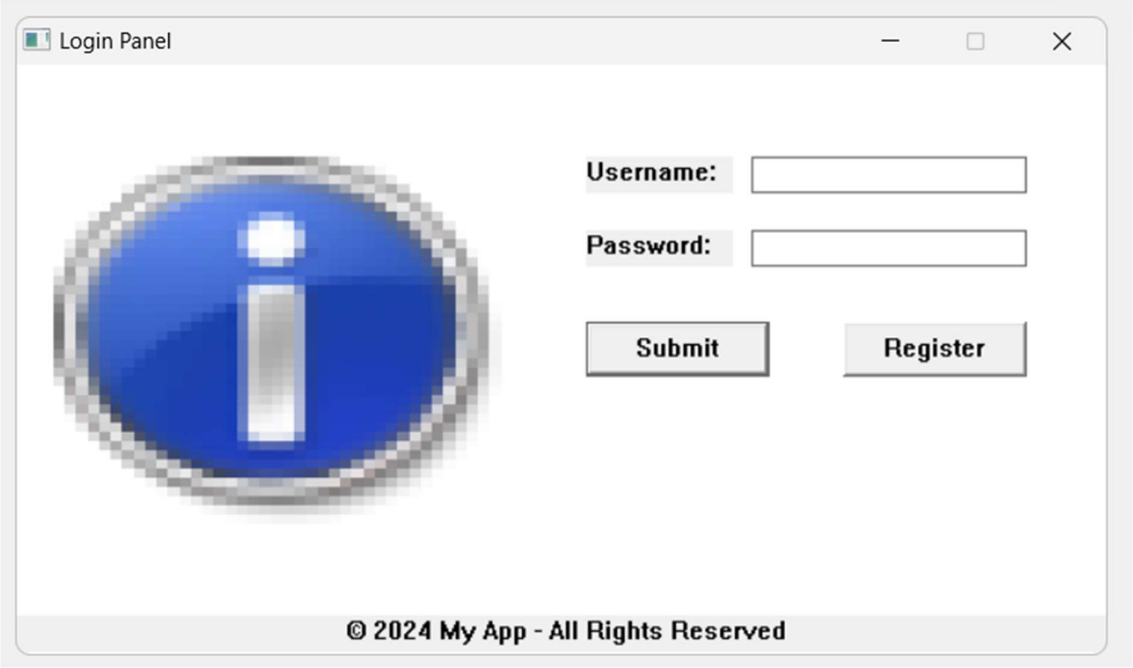


Figure 1: UI Design

Figure 2: Successfully Login

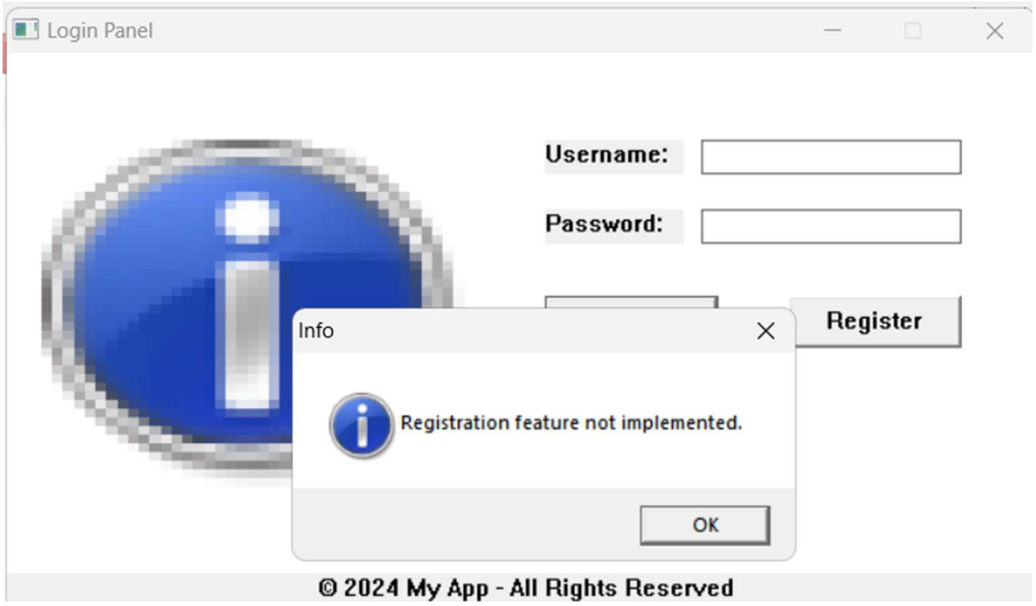


Figure 3: Registration Button clicked.

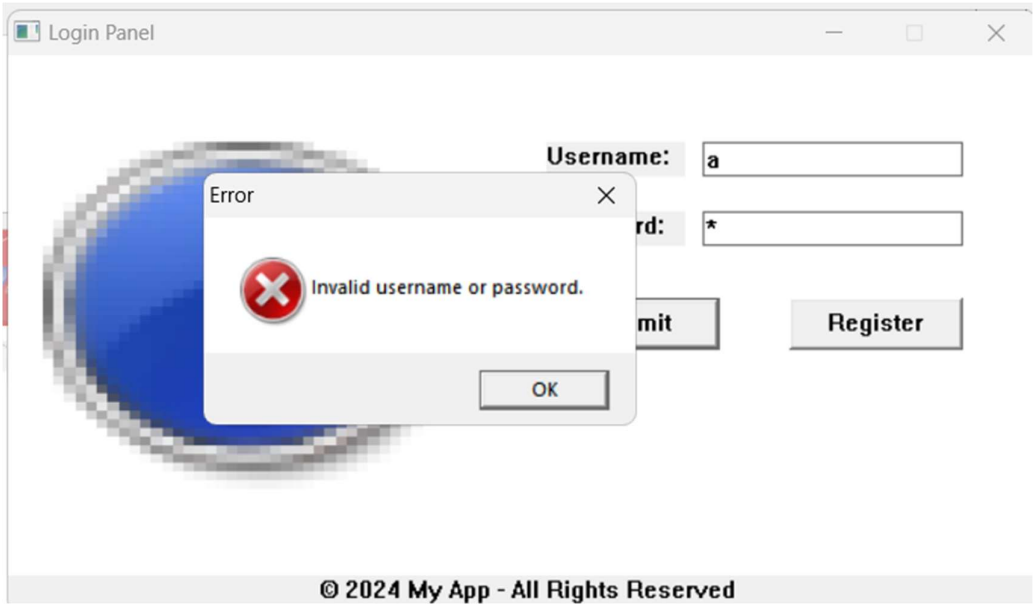


Figure 4: Log In Error

Appendix D

USER MANUAL

Logging In

Follow these steps to log in to the system:

Step 1: Navigate to the Login Page

- When you first launch the application, you will be directed to the **Login Page**.
- On this page, you will see two input fields:
  - **Username Field**: Where you will enter your username.
  - **Password Field**: Where you will enter your password.

These fields allow you to input your login credentials. The username and password must match the stored information for successful authentication.

Step 2: Enter Your Credentials

- **Username**:
  - In the **Username** input field, type **user**.
  - Make sure there are no spaces before or after the username and that the characters are entered exactly as shown (case-sensitive).
  - For example, typing User or USER will result in a login failure because the username is case-sensitive.
- **Password**:
  - In the **Password** input field, type **pass**.
  - As with the username, the password is **case-sensitive**, meaning pass, Pass, or PASS would not work unless typed exactly as pass.
  - The password field will usually hide the typed characters behind asterisks (*) for security purposes. However, the system still records the correct password as you type it.

Step 3: Click "Submit"

- After entering both the **username** and **password**, locate the **Submit** button.
- Click the **Submit** button to process the login attempt.
  - The system will now check if the entered username and password match the stored credentials.

Step 4: Login Outcome

After you click **Submit**, the system will evaluate the credentials:

- **If the entered credentials are correct** (Username: user, Password: pass), the system will display a success message, such as:
  - **"Successfully logged in!"**
  - This message confirms that your login attempt was successful, and the system will grant you access to the next screen or functionality in the application.
- **If the credentials are incorrect**:
  - If either the **username** or **password** is incorrect, the system will display an error message, such as:
    - **"Incorrect username or password."**
  - This indicates that either the username doesn't exist in the system, the password doesn't match, or both. You will then have the option to retry entering your credentials correctly.