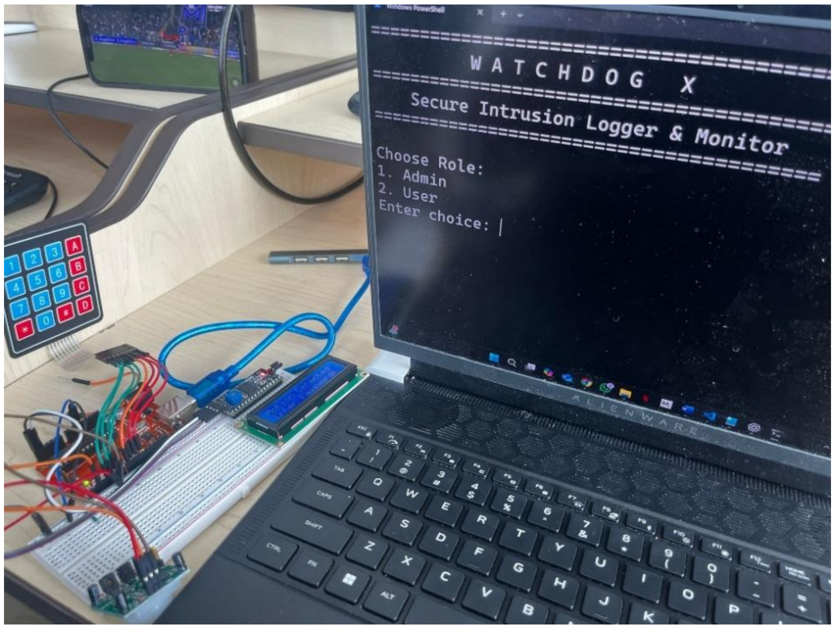
Engineering Project Report

WatchDogX



```
W A T C H D O G   X
Secure Intrusion Logger & Monitor
Choose Role:
1. Admin
2. User
Enter choice: |
```

Group C

Bilal Hussain Mohammed
#9042866
Nikunj Patel
#9047741

Instructor: Safat Khan
Due date:- 14/04/2025

1. Abstract

WatchDogX aims to develop a secure intrusion detection and logging system by integrating hardware sensors with a custom software interface. The primary objective is to detect unauthorized motion using a PIR sensor and alert the user through an audible buzzer. The system lets the alarm deactivate by entering a correct password via a keypad. Once deactivated, the event is sent to the application using serial communication. An LCD gives real-time feedback in this process.

The software component enhances functionality by providing a password-protected interface to manage intrusion logs. It allows real-time logging of intrusion events, as well as manual addition or deletion of log entries. The application ensures that only authorized users can access or modify the records, thereby improving overall system security.
The passwords and logs are safely encrypted.

The project demonstrates a successful hardware-software integration using Arduino and PC communication over a USB serial connection. It serves as a prototype for real-time intrusion monitoring systems and can be extended for use in homes, offices, or sensitive storage areas. The system balances simplicity, reliability, and security within a compact design.

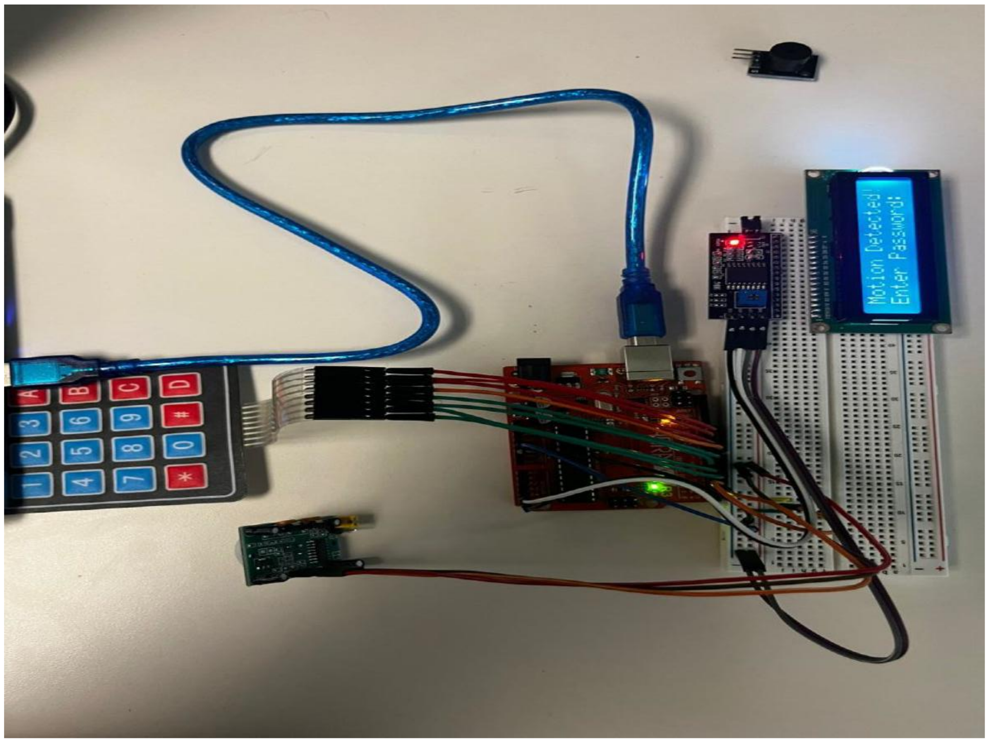
Contents

3. Introduction

3.1 Project Background

Phase 1 – Initial Prototype Development

Security systems are important to protect homes, offices, and storage areas. We started WatchDogX as a simple yet effective intrusion detection system that could detect motion, alert the user, and log suspicious activity. In Phase 1, our goal was to develop a fully functional hardware prototype using accessible and low-cost components.



We assembled a basic circuit consisting of:

- PIR Motion Sensor – to detect movement within the protected area.
- Buzzer – to sound an audible alarm when motion is detected.
- Keypad – to allow the user to enter a passcode and deactivate the alarm.
- 16x2 LCD Display – to display system status, alerts, and prompts for user input.

The system was programmed using Arduino. When the PIR sensor detected motion, the buzzer was triggered, and the LCD displayed a warning message. The user could then enter a predefined password via the keypad to silence the alarm. All motion events, along with timestamps, were sent via serial communication to a connected computer for monitoring and logging. This phase validated the core functionality of WatchDogX, proving that it could reliably detect intrusions, respond with alerts, and allow for basic user interaction. It laid the foundation for further enhancements in both software and hardware in the next phases. After completing Phase 1, we identified areas for improvement. In Phase 2, we focused on enhancing functionality and adding software integration. Key upgrades included:

- Secure log storage using file-based saving instead of just serial output.
- A basic user interface with menu navigation for easier interaction.
- Password protection to restrict access to logs and settings.
- XOR encryption for simple data protection.

These updates made the system more practical, organized, and closer to a real-world security solution.

3.2 Project Objectives

Our overall goal for *WatchDogX* is to create a complete, working intrusion detection and logging system that is both secure and easy to use.

Phase 1 Objectives:

- Detect motion using a PIR sensor.
- Alert the user using a buzzer.
- Allow alarm deactivation with a password.
- Show real-time feedback on an LCD.
- Send log data to the computer using serial communication.

Phase 2 Objectives:

- Store logs using a linked list for flexible, real-time storage.
- Save intrusion logs to files for backup.
- Develop a password-protected software interface for viewing logs.
- Improve data security through encryption and restricted access.
- Optimize the system using free software tools and basic hardware.

3.3 Scope

This project covers the complete development of WatchDogX across two phases.

In Phase 1, we implemented:

- Motion detection using a PIR sensor.
- Audible alerts through a buzzer.
- Password-based alarm deactivation using a keypad.
- Real-time feedback on a 16x2 LCD display.
- Serial communication between Arduino and PC for event logging.

In Phase 2, we enhanced the system with:

- Log storage using a linked list for efficient real-time data handling.
- Secure file storage of intrusion logs on the computer.
- A basic user interface to view, add, or delete logs.
- Admin-only access and password authentication within the UI.
- Encrypted log data and secure handling to prevent unauthorized access

3.4 Overview

This report is structured to cover all aspects of the project in a logical sequence:

- **Introduction**
  Provides background, objectives, and scope of the WatchDogX system.
- **Project Requirements**
  Lists functional, non-functional, and technical requirements the system must meet.
- **Materials and Components**
  Details hardware components, cost analysis, and software tools used in the project.
- **Software Requirements Specification (SRS)**
  Describes the overall system design, external interfaces, user needs, and project constraints.
- **Software Design Description (SDD)**
  Explains the architecture, data handling, and module-level design of the application.
- **Circuit Design**
  Includes wiring diagrams and descriptions for hardware setup and sensor integration.
- **Implementation**
  Covers the structure of the code, core functionalities, key code snippets, and problem-solving approaches.
- **Testing and Results**
  Shows how each component was tested with expected vs. actual outputs, and documents system reliability.
- **Discussion**
  Analyzes the results, explains limitations, and evaluates system performance during the demo.
- **Conclusion & Future Work**
  Summarizes achievements and proposes possible improvements like mobile integration and wireless communication.
- **References & Appendices**
  Contains all cited sources, full code listings, diagrams, and additional supporting materials.

4. Project Requirements

4.1 Functional Requirements

The WatchDogX system must support the following core functionalities:

- **Sensor Integration**
    - Detect motion using a PIR sensor.
    - Send sensor data to the software for further processing.
- **Data Display**
    - Display messages and status updates on an LCD module.
- **Alarm System**
    - Activate a buzzer when intrusion is detected.
    - Allow deactivation using a password via keypad input.
- **Data Logging and Storage**
    - Save all user-generated and sensor-related data to disk.
    - Use different files for logs, configurations, and access history.
    - Track login times, user activities, and event timestamps.
- **User Interface**
    - Provide a Command Line Interface (CLI) for navigation and interaction.
    - Allow users to view logs, live data, and manage information.
    - Enable registration of user profiles with different access levels.
    - Include an administrator account with full control.
    - Support a safe shutdown option via the interface.
- **Security Features**
    - Encrypt all stored data (plain text files not allowed).
    - Password protection for both login and access to sensitive operations.
- **Linked List Integration**
    - Use linked lists for managing dynamic data like logs or user sessions.
    - Convert linked list code into a reusable header-based library.
- **Serial Communication**
    - Ensure reliable serial communication with the Arduino Uno R3.
    - Use a trimmed and customized serial library for this project.

4.2 Non-Functional Requirements

- **Performance**
  - The system should respond to sensor triggers and update the display within 1 second.
  - Sensor polling frequency should be optimized for real-time detection (e.g., every 1–2 seconds).
- **Security & Data Protection**
  - Encrypted storage is mandatory to prevent tampering or unauthorized access.
  - Password mechanisms must be strong and secure.
- **Reliability & Stability**
  - The system should run continuously without crashes.
  - File handling and user session management must be error-free.
- **Usability**
  - The CLI must be user-friendly, clearly formatted, and easy to navigate.
  - Clear separation of user and admin roles to ensure secure operation.

4.3 Technical Requirements

Category	Requirement Description
Platform Requirements	Must operate on an Arduino Uno, with power supply options (USB or battery).
Interface Requirements	MUST interface with the output component (LCD) for feedback.
Sensor Integration	Utilize data from a minimum of 2 sensors from the Elegoo kit in a meaningful way.
User Input Requirements	User input is used to control or adjust sensor parameters (e.g., potentiometer).
Calculation Requirements	Include calculations related to sensor data, if applicable (e.g., averages, thresholds).
Output Requirements	Display meaningful results based on sensor data (e.g., visual, auditory, or data storage).
Documentation Provided	Provide basic documentation: setup steps, wiring, and sample code for the project.

5. Materials and Components

5.1 Hardware Components

Component		**Description**
Arduino UNO R3		Acts as the brain of the system, managing inputs, outputs, and communication with the laptop.
Breadboard		Detects human movement to identify potential intrusions.
LCD		Displays system messages, such as prompts and status updates.
4x4 keypad		Allows the user to input a password to deactivate the alarm.
Buzzer		Produces a loud sound to alert when motion is detected.
Jumper Wires		Connect all components to the Arduino on the breadboard for prototyping.
I2C Adapter		Simplifies LCD wiring and frees up Arduino pins by enabling two-wire communication.
Laptop		Logs and manages intrusion data with password protection via serial communication.

5.2 Cost Analysis

Component	Quantity	Unit Price (USD)	Total Cost (USD)
Arduino UNO R3	1	$10.00	$10.00
PIR Motion Sensor	1	$3.50	$3.50
Buzzer (Active)	1	$1.00	$1.00
Red LED	1	$0.10	$0.10
Breadboard (Half-size)	1	$3.00	$3.00
Jumper Wires (Pack)	1	$2.00	$2.00
I2C LCD Adapter	1	$1.50	$1.50
Total Estimated Cost			$21.10

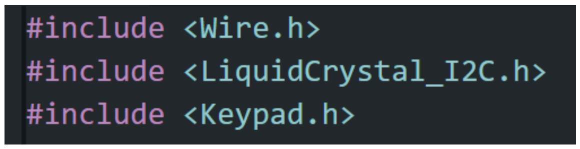
Note: This project needs a system to run which must have good specifications. The laptop can cost around $500-$1500 depending on the user. This cost is not included in the table.

5.3 Software Tools

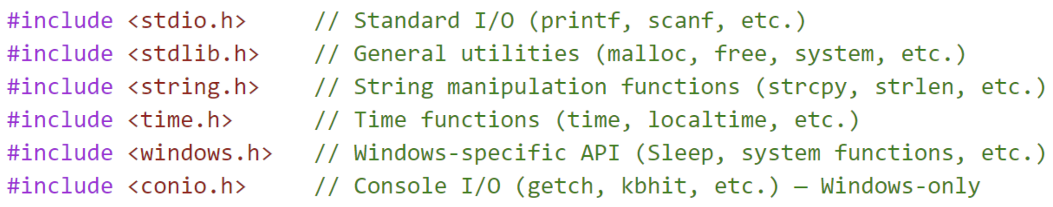
- **IDE**: Arduino IDE (v1.8.19)
  - Used for writing, compiling, and uploading code to the Arduino UNO board. It provides built-in serial monitor functionality to test communication between the board and laptop.

- **VS Code (Visual Studio Code):**
  -Used for developing the software application that receives and logs intrusion data from the Arduino over serial communication.

- **Libraries**:
  - LiquidCrystal_I2C.h – For controlling the 16x2 LCD using the I2C adapter.
  - Wire.h – Enables I2C communication between the Arduino and the LCD module.
  - Keypad.h – Handles multi-key input from the 4x4 matrix keypad.



```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Keypad.h>
```

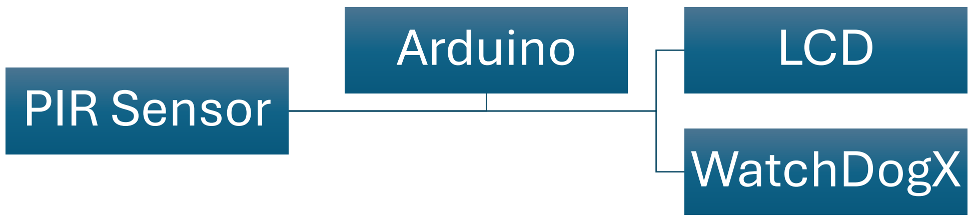


```
#include <stdio.h>     // Standard I/O (printf, scanf, etc.)
#include <stdlib.h>    // General utilities (malloc, free, system, etc.)
#include <string.h>    // String manipulation functions (strcpy, strlen, etc.)
#include <time.h>      // Time functions (time, localtime, etc.)
#include <windows.h>   // Windows-specific API (Sleep, system functions, etc.)
#include <conio.h>     // Console I/O (getch, kbhit, etc.) — Windows-only
```

- **Other Tools**:
  TinkerCad (Autodesk, Web Version, Accessed March 2025): Used for circuit simulation and testing before building the physical hardware.

6. Software Requirements Specification (SRS)

Project:	
Document:	Software Requirements Specification
Author:	Bilal Hussain Mohammed
Date:	28-03-2025
Version:	1

Overall Description	
Background:	In today's world, security threats such as unauthorized access, theft, and intrusion are increasing in both residential and commercial spaces. Traditional surveillance systems often lack real-time response, secure logging, and user-specific access control. WatchDogX addresses this gap by providing a responsive and user-friendly intrusion detection and logging system. It enhances security by combining motion detection hardware with a password-protected software application that logs events, sounds alerts, and maintains encrypted records for future review.
Project Overview:	WatchDogX is a security-focused software application designed to monitor and log intrusion events. It interfaces with hardware sensors to detect motion, provides real-time alerts, and allows secure access to log data through a user-friendly interface.
Block Diagram:	

Overall Description	
Block Descriptions:	➢ **PIR Sensor**: Detects motion by sensing infrared radiation from moving objects (like humans) and sends a signal to the Arduino. ➢ **Arduino**: Acts as the main controller. It receives signals from the PIR sensor, activates the buzzer and LCD, and sends intrusion data to the WatchDogX software via serial communication. ➢ **LCD**: Displays system messages such as "Intruder Detected", "System Armed", or "Access Granted", giving visual feedback at the hardware level. ➢ **WatchDogX**: Software application running on a PC. It receives intrusion alerts from the Arduino, logs the events securely, provides password-protected access, and displays real-time logs to the user.
Typical User:	➢ A typical user of WatchDogX is someone responsible for monitoring a secured area, such as a homeowner, office administrator, or security personnel. They are expected to have basic computer literacy and understand simple user interfaces to interact with the system effectively.
Constraints:	➢ **Hardware Limitations**: Arduino has limited memory and processing power, restricting advanced features on the hardware side. ➢ **Serial Communication Dependency**: The software relies on a stable serial connection between the Arduino and the PC. ➢ **User Privileges**: Installation and full access may require administrative rights on the host system. ➢ **Platform Compatibility**: WatchDogX is developed specifically for Windows and may not run on other operating systems without modification. ➢ **Storage Restrictions**: Log files must be securely stored, which may be limited by disk space or permissions. ➢ **Real-time Processing**: Ensuring low-latency communication and timely alerts requires optimized code and an uninterrupted power supply.

Specific Requirements

External Interfaces:

Sources of Input and Output:

Type	Source / Destination	Description
Input	PIR Sensor (via Arduino)	Detects motion and triggers the Arduino.
Input	Arduino (via Serial Communication)	Sends intrusion alert signals to the WatchDogX software.
Input	User Input (GUI)	Login credentials, log view requests, delete actions, and password changes.
Input	Encrypted Log File (Optional)	Decrypted and loaded on application startup if available.
Output	LCD (via Arduino)	Displays intrusion or status messages like “Intruder Detected”.
Output	Buzzer (via Arduino)	Activates alarm in case of motion detection.
Output	User Interface (PC)	Displays logs, alerts, and menu options.
Output	Encrypted Log File	Stores intrusion events securely on the local system.
Output	Popup Messages (GUI)	Provides alerts for login status, actions taken, and system errors/successes.

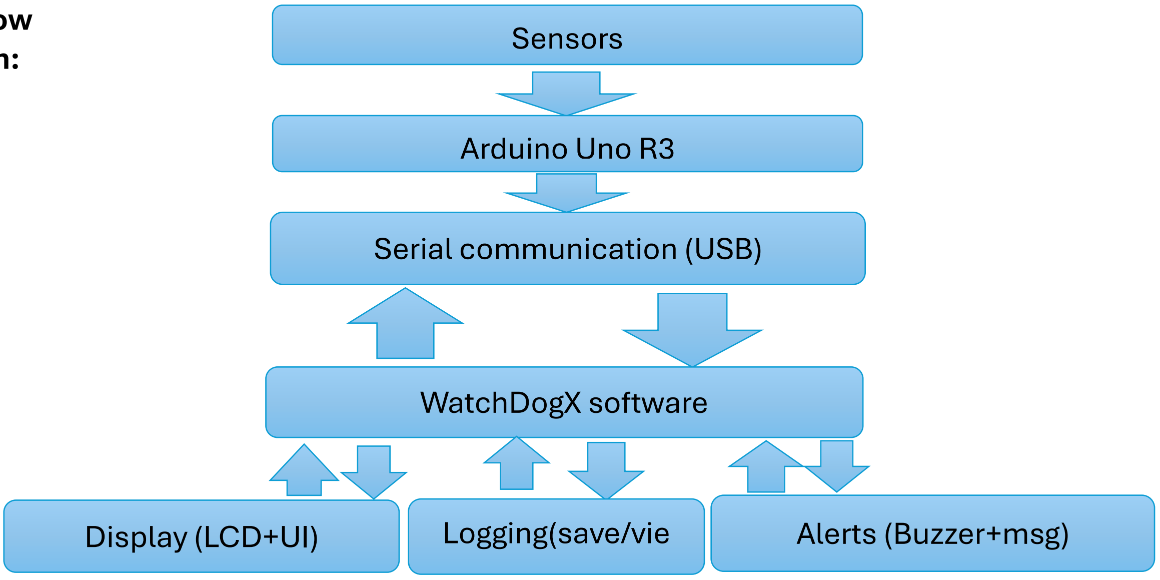
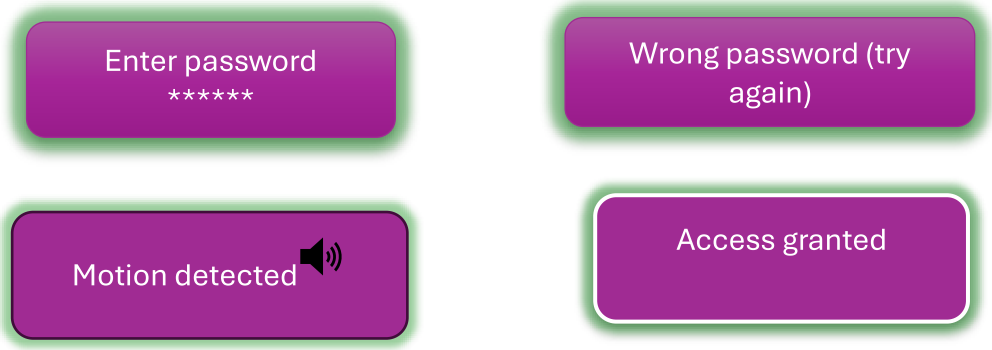
Specific Requirements:

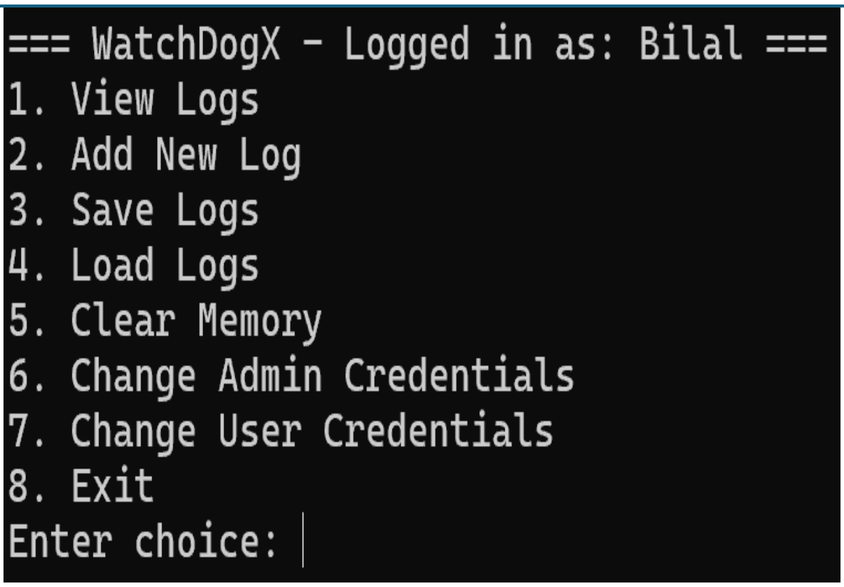
Number	Requirement	MuSCoW
1	platform requirements (PC, MAC, web-based, etc.)	**Must**
2	interface requirements (command line, interactive text, GUI, etc.)	**Must**
3	user I/P requirements (acceptable ranges for user-specified values, and defaults where applicable)	**Must**
4	calculation requirements (accuracy, precision of calculations, use of random numbers, etc.)	**Should**
5	visual O/P requirements (precision of displayed values, layout of tables, etc.)	**Must**
6	storage/control O/P requirements (local/cloud storage, devices to be controlled, etc.)	**Must**
7	documentation provided (manuals, instructions, etc.)	**Should**

7. Software Design Description (SDD)

Project:	
Document:	Software Design Document
Author:	Nikunj Patel
Date:	29-3-2025
Version:	1.0

System Description	
System Description:	WatchDogX is a secure intrusion detection and logging system that uses hardware sensors to detect motion, raise alerts, and store event logs. It integrates with a password-protected software interface for secure data management, allowing users to monitor events, review logs, and manage data efficiently.
Development Platform:	• **Hardware:** Arduino Uno R3, PIR Sensor, Buzzer, LCD Display, 4x4 Keypad, I2C Adapter • **Software:** Arduino IDE, Visual Studio Code, Tinkercad (for circuit simulation) • **Programming Language:** C • **Libraries:** LiquidCrystal_I2C, Wire, Keypad, DHT, EEPROM
Execution Platform:	• **Hardware:** Laptop with Windows 10 or higher, USB Connection to Arduino • **Software:** WatchDogX Application (custom interface) • **Communication:** Serial Communication via USB

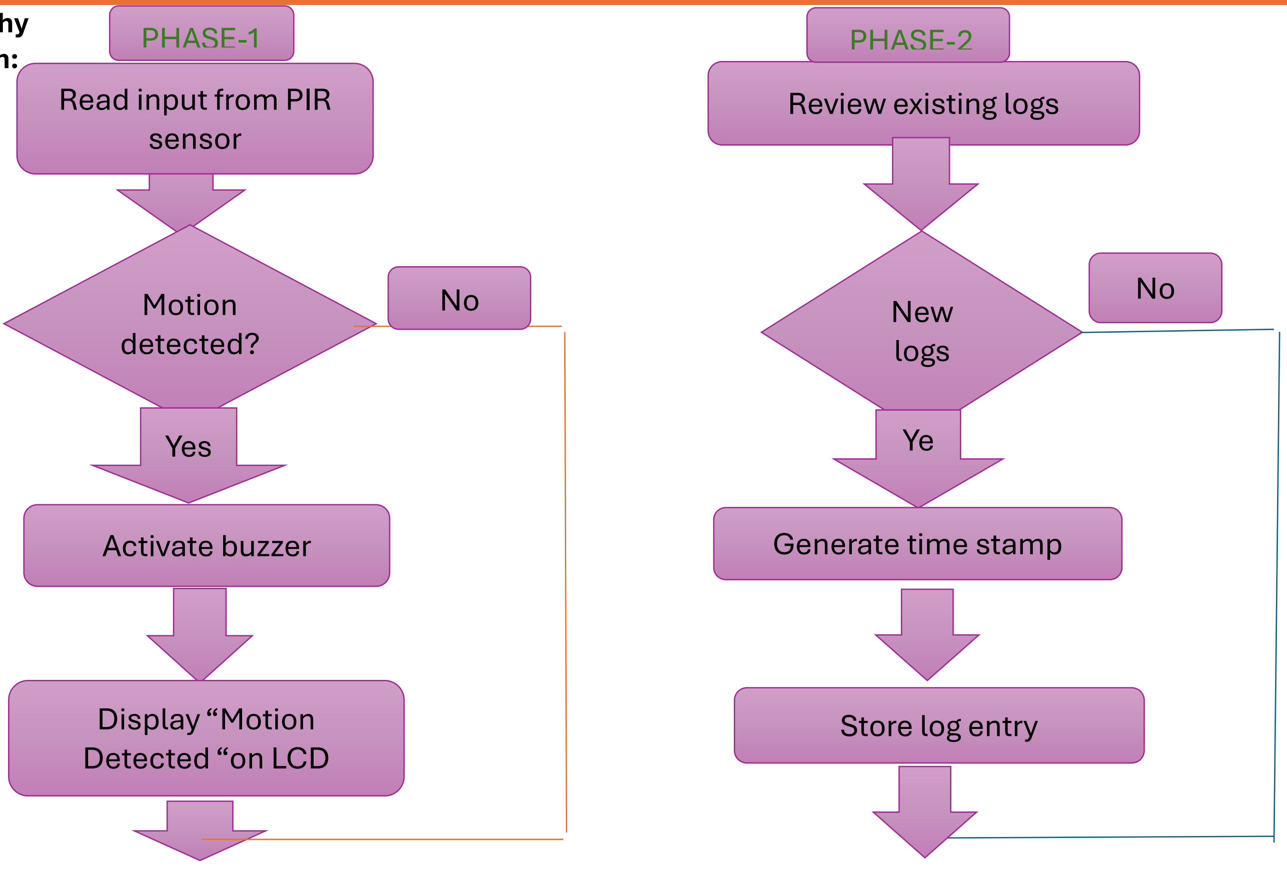
Data Flow Diagram:	
System Interfaces:	



```
=== WatchDogX - Logged in as: Bilal ===
1. View Logs
2. Add New Log
3. Save Logs
4. Load Logs
5. Clear Memory
6. Change Admin Credentials
7. Change User Credentials
8. Exit
Enter choice:
```

Detailed Design

Hierarchy Diagram:



Modules

1)Module Name:	Detect motion
Purpose:	Continuously monitor the PIR sensor to detect any movement and trigger the system response.
Input:	PIR sensor signal (HIGH/LOW)
Output:	Alert trigger (buzzer ON) LCD message ("Motion Detected") Motion log entry

2)Module Name: Validate password

Purpose: Accept and verify user-entered password to either grant or deny access.

Input:
- Keypad input (user password)
- Stored password (EEPROM)

Output:
- Access status (granted or denied)
- LCD message ("Access Granted" / "Wrong Password")
- Buzzer OFF (if access granted)
- Login attempt log

3)Module name: Log events

Purpose: Record every significant system event (motion detection, access, error) with timestamp.

Input:
- Event type (motion, login, error)
- Event timestamp

Output:
- Log entry saved to EEPROM
- Log sent to PC software (via serial)

4) Module name: Trigger alerts

Purpose: Activate visual or audible alerts based on motion detection or failed login.

Input: Event type (e.g., motion detected, wrong password)
Output:
- Buzzer ON/OFF

- LCD alert message

5)Module name: Display message

Purpose: Show system status, alerts, and errors on the LCD.

Input: Message text

Output:

- Text displayed on 16x2 LCD

6)Module name: Handle error

Purpose: Identify and respond to hardware or communication failures.

Input: Sensor signal (disconnected/null)

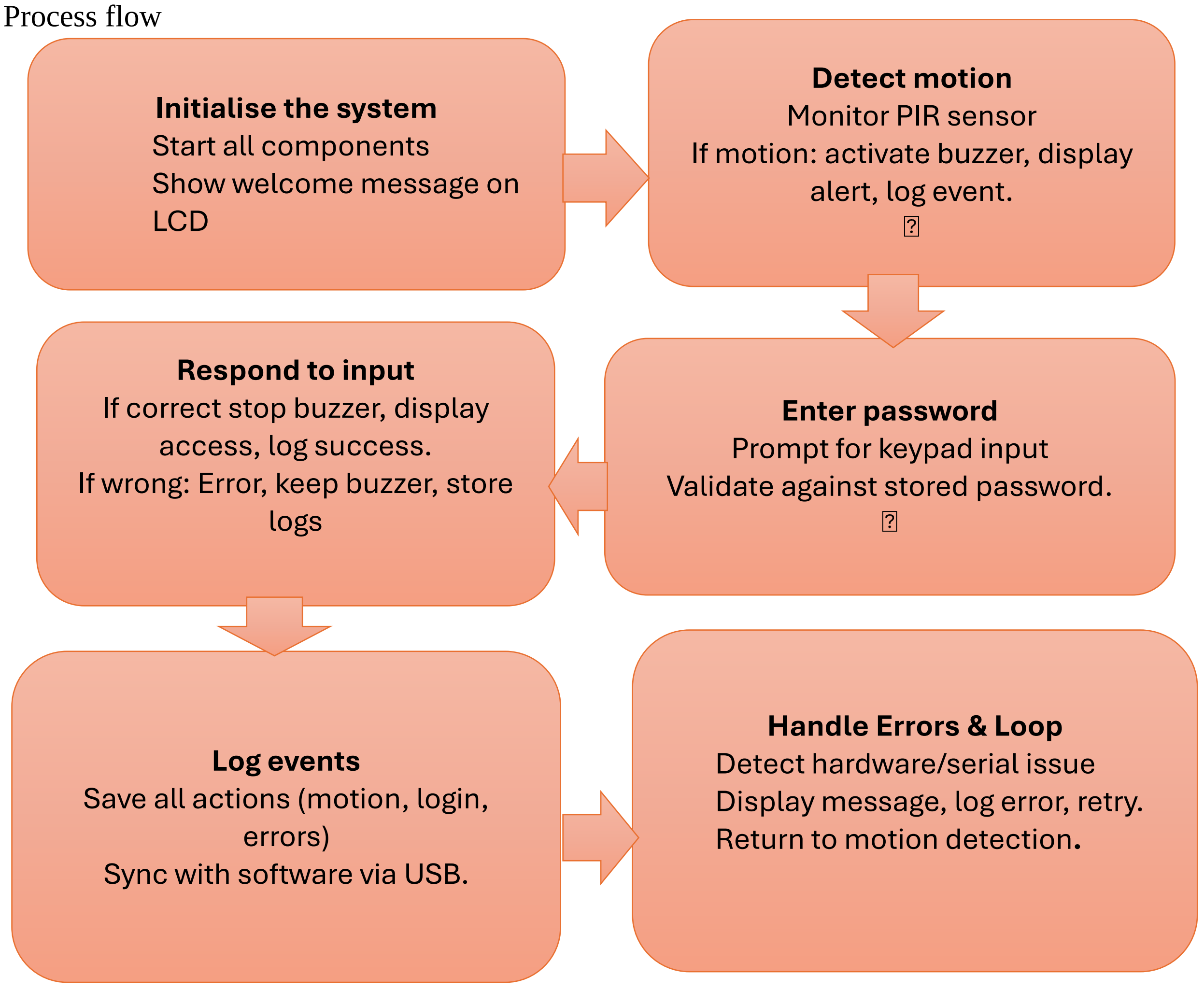
- Serial status
- Power status

Output:

- LCD error message
- Logged error entry
- Retry attempt or system reset

Algorithm:

Process flow


Initialise the system
Start all components
Show welcome message on LCD
Detect motion
Monitor PIR sensor
If motion: activate buzzer, display alert, log event.
Enter password
Prompt for keypad input
Validate against stored password.
Respond to input
If correct stop buzzer, display access, log success.
If wrong: Error, keep buzzer, store logs
Log events
Save all actions (motion, login, errors)
Sync with software via USB.
Handle Errors & Loop
Detect hardware/serial issue
Display message, log error, retry.
Return to motion detection.

Test values

Reliability Test:

Test scenario	Input	Expected output
Motion Detection	Movement	Motion detected on LCD, Buzzer on
No motion	No movement	No alert displayed
Correct password entry	Correct password	Access granted, buzzer off
Incorrect password entry	Wrong password	Wrong password, buzzer on

Robustness Test:

Test scenario	Input	Expected output
Empty password entry	No input	Invalid password
Exceeding password length	Over 8 characters	Password length error
Motion sensor failure	No sensor response	Sensor error.

Exceptions Test:

Exception cause	Possible effect	Handling method
Sensor disconnection	No motion detected	Display sensor error on LCD
Power failure	System shutdown	Data is backed up in logs
Serial communication error	No data transfer	Retry connection with error log

8. Circuit Design

8.1 Circuit Diagram

- Figure 1 shows the connections of the components. This was generated by using TinkerCad.

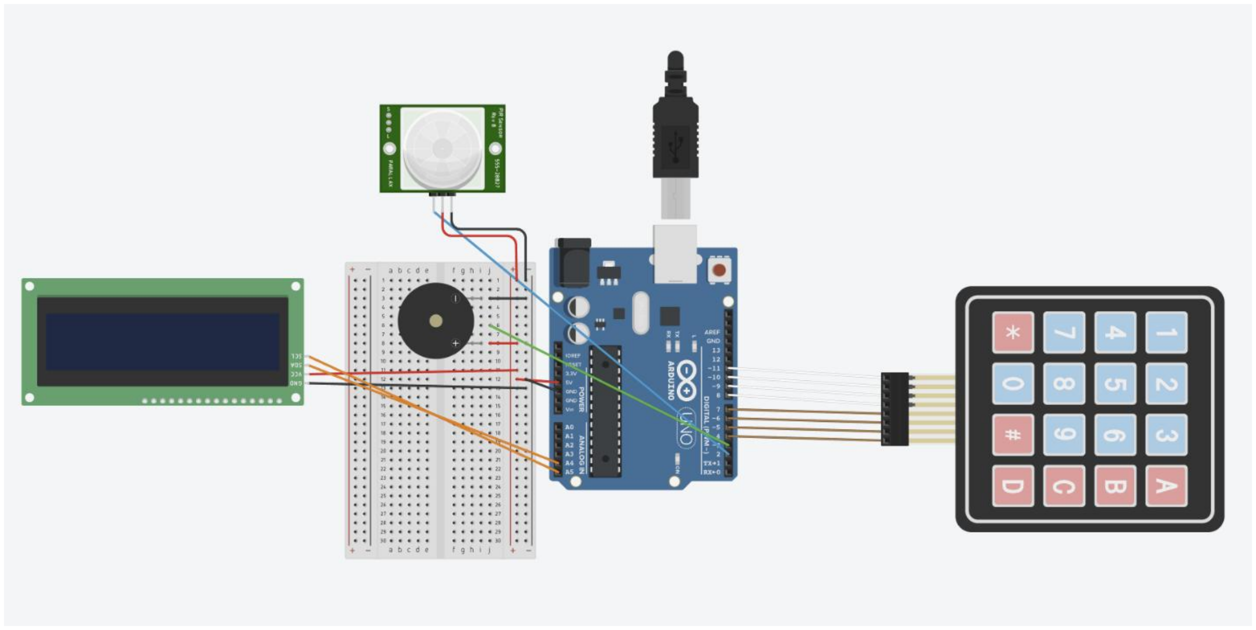


Figure 1: TinkerCad diagram for wiring and components placements.

Schematic View of the Diagram:

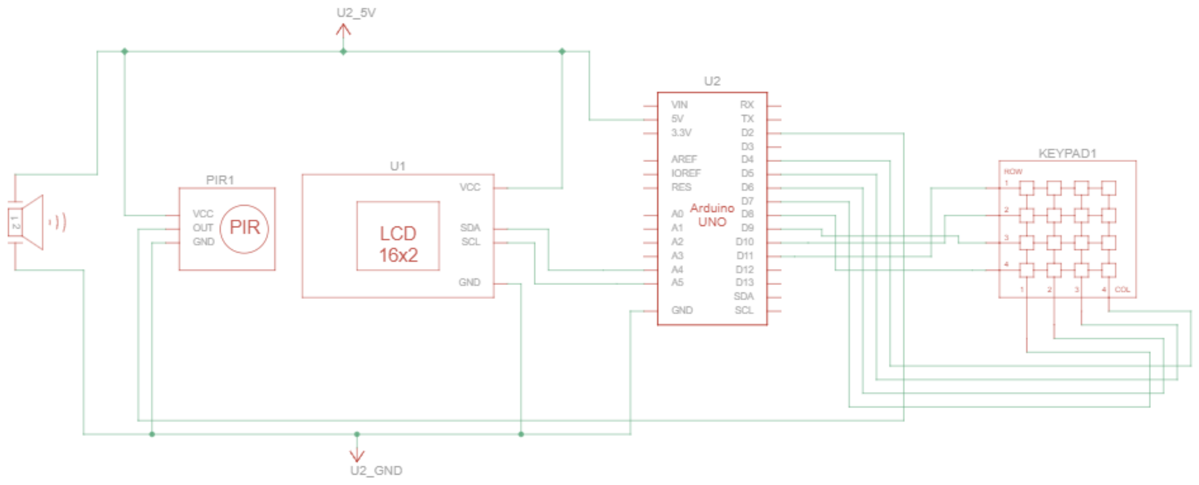


Figure 2 This is the Schematic View of the same Circuit extracted from TinkerCad as well.

8.2 Wiring Description

Component	Arduino Pin	Component Pin	Description
HC-SR501 PIR Sensor	Digital Pin 7	OUT	Detects motion and sends a signal to Arduino.
	5V	VCC	Powers the PIR sensor.
	GND	GND	Ground connection for the sensor.
Buzzer	Digital Pin 9	Positive (+)	Emits sound when motion is detected.
	GND	Negative (-)	Ground connection for the buzzer.
16x2 LCD Display	SDA	A4	Register select pin for LCD control.
	SCL	A5	Analog pin for LCD.
	5V	VCC	Powers the LCD.
	GND	GND	Ground connection for the LCD.
Keypad (4x4)	Digital Pins 2–9	R1–R4, C1–C4	Used to input passwords and interact with the system.

9. Implementation

9.1 Code Structure

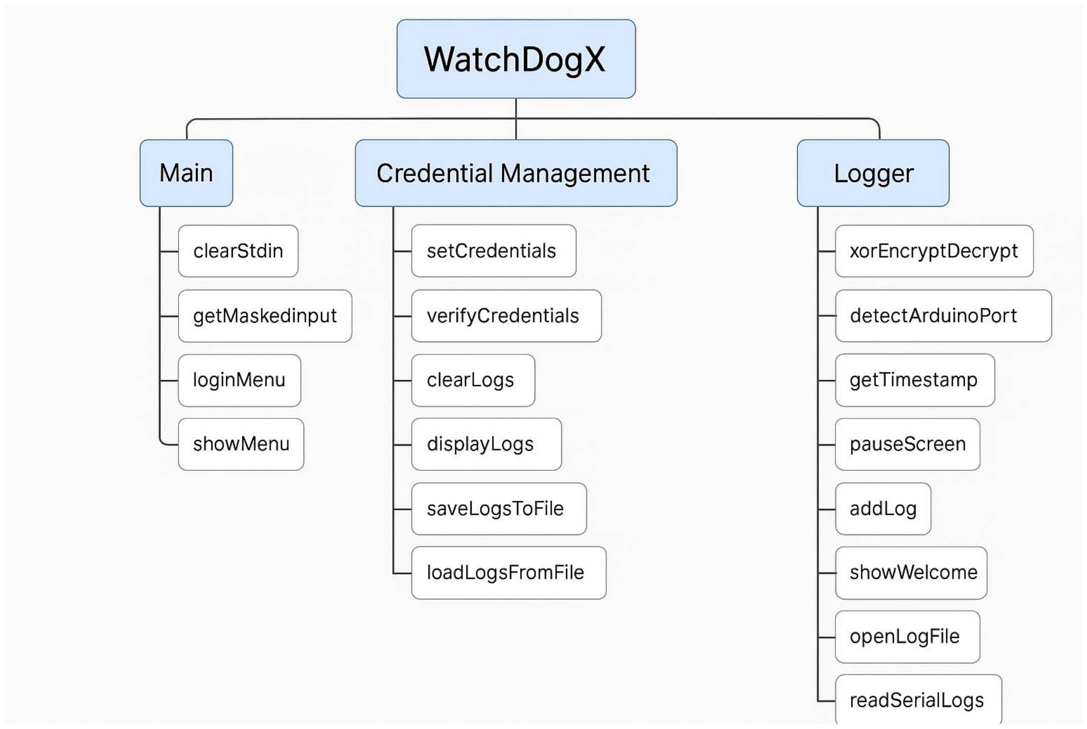
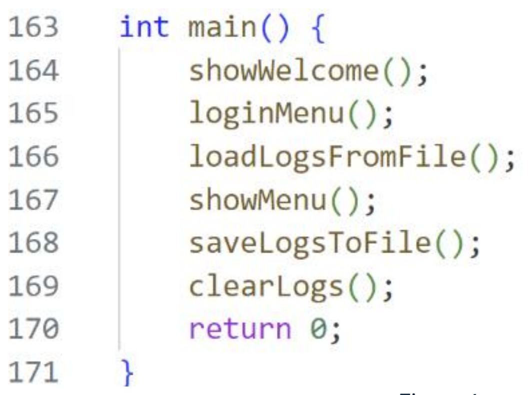


Figure 3: This is a Diagram of the Code Structure made by using Draw.io[17].

The diagram above illustrates the full code structure of the WatchDogX system, including its main logic, credential management, logging system, and Arduino integration functions.

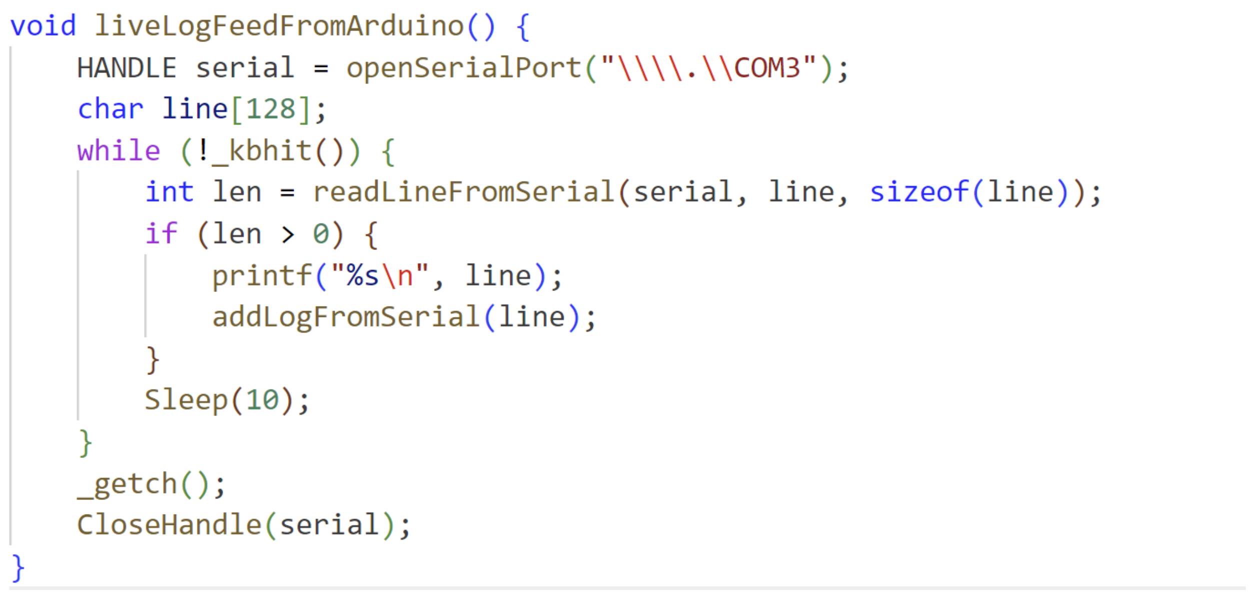
9.2 Key Code Snippets



```
int main() {
    showWelcome();
    loginMenu();
    loadLogsFromFile();
    showMenu();
    saveLogsToFile();
    clearLogs();
    return 0;
}
```

Figure 4

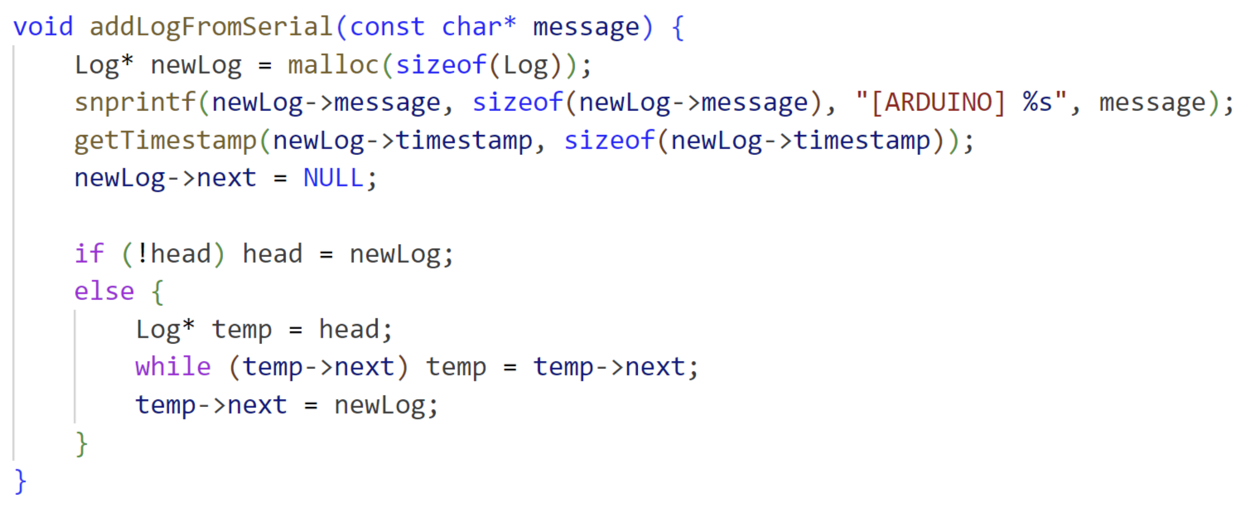
As shown in Figure 4:-The main() function is the entry point of the WatchDogX program. It initializes the system by displaying the welcome screen, handling user login, and loading saved logs. After the user interacts with the menu, it saves any new logs and clears memory before exiting.



```c
void liveLogFeedFromArduino() {
    HANDLE serial = openSerialPort("\\\\.\\COM3");
    char line[128];
    while (!_kbhit()) {
        int len = readLineFromSerial(serial, line, sizeof(line));
        if (len > 0) {
            printf("%s\n", line);
            addLogFromSerial(line);
        }
        Sleep(10);
    }
    _getch();
    CloseHandle(serial);
}
```

Figure 5

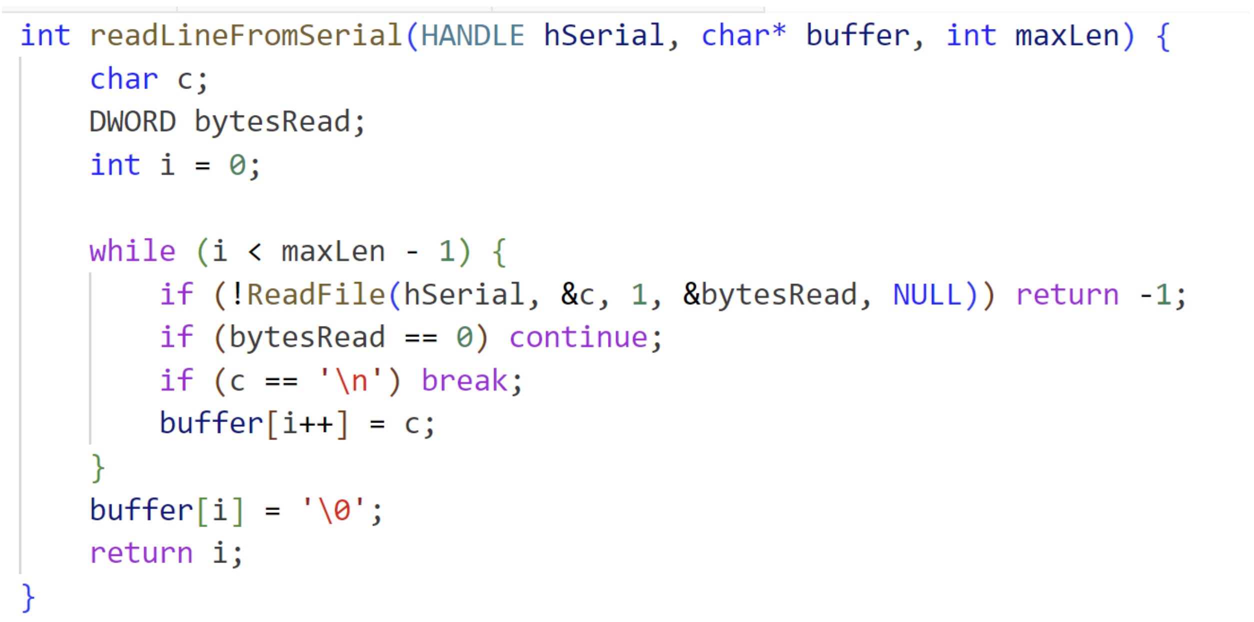
As shown in Figure 5:
This function allows the admin to view real-time logs from the Arduino. It continuously reads lines until a key is pressed, displaying them on screen and storing them into the WatchDogX memory.-



```c
void addLogFromSerial(const char* message) {
    Log* newLog = malloc(sizeof(Log));
    snprintf(newLog->message, sizeof(newLog->message), "[ARDUINO] %s", message);
    getTimestamp(newLog->timestamp, sizeof(newLog->timestamp));
    newLog->next = NULL;

    if (!head) head = newLog;
    else {
        Log* temp = head;
        while (temp->next) temp = temp->next;
        temp->next = newLog;
    }
}
```

Figure 6

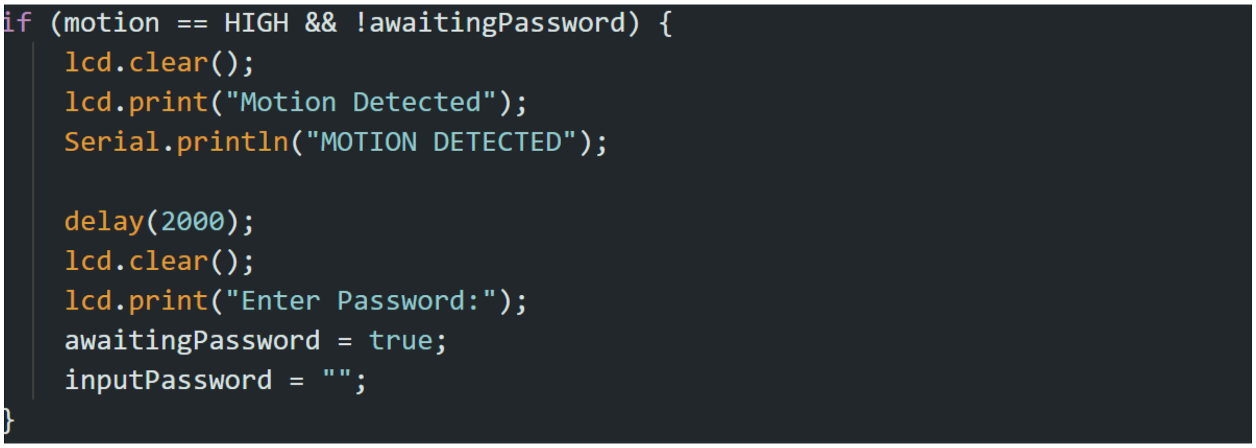
As shown in Figure 5:-
This code creates a new log entry in WatchDogX from Arduino serial data. Each log is timestamped and stored in memory for later viewing or saving.



```c
int readLineFromSerial(HANDLE hSerial, char* buffer, int maxLen) {
    char c;
    DWORD bytesRead;
    int i = 0;

    while (i < maxLen - 1) {
        if (!ReadFile(hSerial, &c, 1, &bytesRead, NULL)) return -1;
        if (bytesRead == 0) continue;
        if (c == '\n') break;
        buffer[i++] = c;
    }
    buffer[i] = '\0';
    return i;
}
```

Figure 7

As shown in Figure 6:-
This function reads one complete line from the serial port. Each message sent from the Arduino (like motion alerts or password attempts) ends in a newline character (\n) and is captured here.



```
if (motion == HIGH && !awaitingPassword) {
    lcd.clear();
    lcd.print("Motion Detected");
    Serial.println("MOTION DETECTED");

    delay(2000);
    lcd.clear();
    lcd.print("Enter Password:");
    awaitingPassword = true;
    inputPassword = "";
}
```

Figure 8

As shown in Figure 7:-
When the PIR sensor detects motion, the LCD displays a message and the system sends "MOTION DETECTED" via serial to the PC. It then asks the user to enter a password.

9.3 Challenges and Solutions

While working on WatchDogX, we faced a few real challenges that took time and testing to fix.

1. **Serial communication problem:**
   Our Arduino was detecting motion correctly. But the data wasn't showing up in our C program on the PC. The Arduino and PC were technically connected, but the C code wasn't reading the data properly. After checking everything we found out it was a problem with the code, baud rate and how data was being read. We fixed it by adjusting the setting and adding some delay and flushing in the C code so the data could be read clearly.

2. **Linking Motion Detection with the C program:**
   Even after motion was detected, getting that to trigger a proper response in the C interface was difficult. The data was coming in, but our code didn't know how to react to it. We solved this by setting up specific signals in the code so that when motion detected, the C program could respond right away. Like showing a message or saving event.

3. **Storing Motion Data with Linked lists:**
   We wanted to keep a log of each motion detection event, so we used a linked list in C to store them. Since we didn't know how many events there would be, linked lists were useful. It was tricky at first because of memory issues, but we made sure to handle memory properly and clear out old data when needed.

4. **Keeping the Data Secure:**
   Since this project involved security. We didn't want anyone to see or change the logs. So, we added basic encryption to the data when sending it to the PC and added a login system in our program to protect access.

   These issues made the project challenging but solving them helped us understand how real systems work and how hardware and software need to communicate smoothly.

10. Testing and Results

10.1 Testing Plan

In this section, we describe how we tested the WatchDogX system to check if it works as expected. The main goal of testing was to ensure that all the parts of the system, including the PIR sensor, buzzer, keypad, LCD display, and software, function correctly.

We performed different tests to see how well the system detects motion, activates alarms, and handles user input. Additionally, we tested if the system can securely store intrusion logs, display accurate messages, and communicate smoothly with the software using serial communication.

By simulating real-world scenarios, such as motion detection, password entry, and log management, we measured how quickly and accurately the system responded. The results of these tests helped us identify any issues and make improvements where needed.

The following sections explain the specific tests we conducted, the results we observed, and how well the system met our expectations.

10.2 Testing Phase 1

- **PIR Sensor Functionality:**

We tested the PIR (Passive Infrared) sensor to check if it could detect motion and respond correctly. The goal was to see if it could detect movement, turn on the buzzer, and show a warning message on the LCD screen.

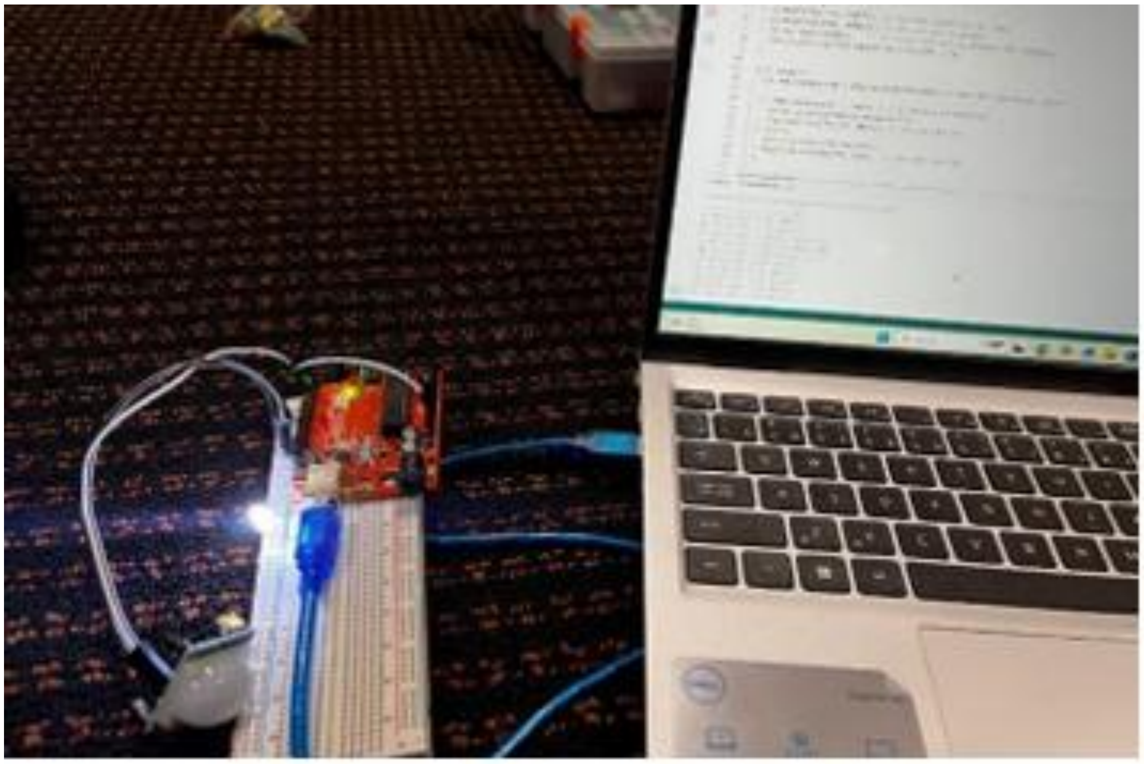
What We Planned: We wanted to confirm that the sensor would detect motion within its range, trigger the alarm, and display the correct message. We also planned to check how fast it responded and whether it gave false alarms when there was no actual motion.

What We Thought: We expected the PIR sensor to quickly detect motion and activate the buzzer and LCD. We also thought it might sometimes detect false movements caused by things like pets, temperature changes, or shadows.

What We Used

- PIR Sensor (HC-SR501) – To detect motion
- Arduino Uno R3 – To control the sensor and other components
- Buzzer and LCD – To sound an alarm and show messages
- Keypad – For entering the password to stop the alarm
- Laptop with Arduino IDE – For writing and uploading code

What We Tested as shown in figure below:



1. Motion Detection Test: Checked if the sensor detected motion and triggered the buzzer and LCD.
2. No Motion Test: Made sure the sensor stayed inactive when no motion was detected.
3. False Trigger Test: Observed if the sensor falsely detected motion from environmental changes.
4. Response Time Test: Measured how quickly the system reacted after motion was detected.
5. Range and Angle Test: Tested how far and at what angle the sensor could detect motion.

- **Keypad and Password Input**
  We tested the keypad and password input to ensure it worked properly for turning off the alarm. The main goal was to check if the system accepted the correct password, showed clear messages, and handled wrong attempts correctly.

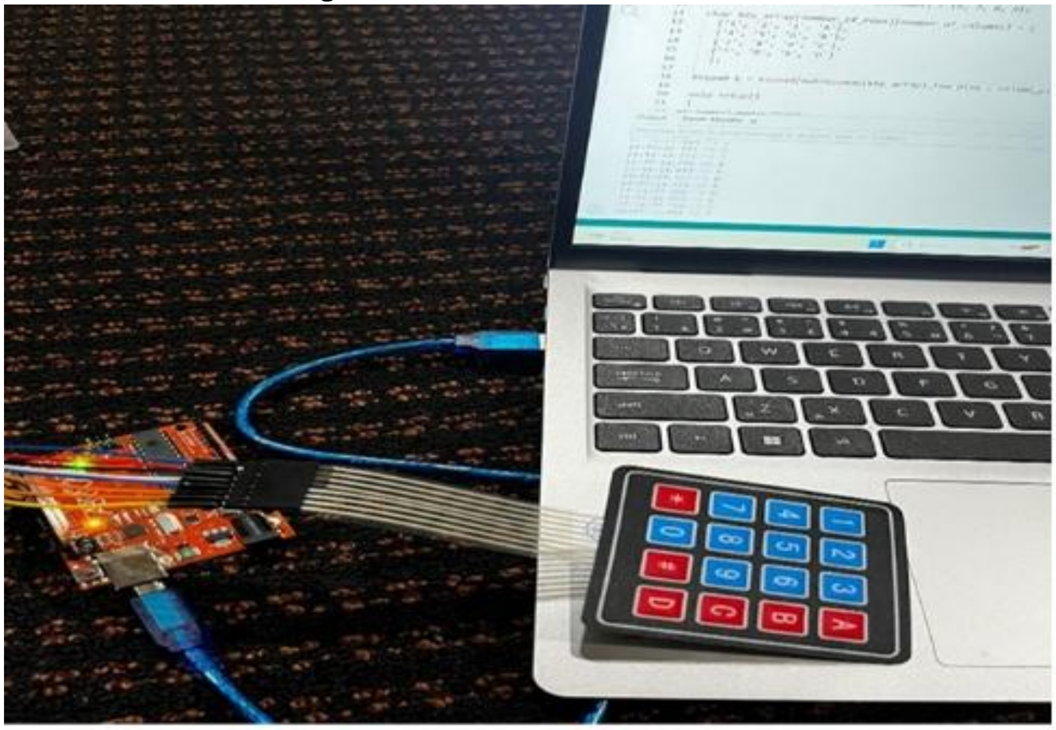
What We Planned: We wanted to see if the keypad could take input without errors, confirm the correct password, and stop the alarm. It was also important to check how the system responded to wrong passwords and displayed messages on the LCD.

What We Thought: We expected the system to display "Enter Password" on the LCD when motion was detected. If the correct password was entered, the alarm would turn off, and the display would confirm it. Wrong passwords would trigger a "Wrong Password" message and keep the alarm active.

What We Used:

- **4x4 Matrix Keypad** – For entering the password
- **Arduino Uno R3** – For processing the input
- **Buzzer and LCD** – For alerts and feedback
- **Laptop with Arduino IDE** – For testing and monitoring

What We Tested as in figure below:



- Correct Password Test – To check if the system turned off the alarm when the correct password was entered.
- Wrong Password Test – To ensure the alarm stayed active and showed an error message.
- Multiple Attempts Test – To see how the system handled repeated wrong attempts.
- Display Test – To confirm clear and correct messages on the LCD.
- Response Time Test – To measure how quickly the system responded to password input.
- These tests helped us ensure the keypad and password input system worked as expected.

- **LCD Display Output**
  We tested the LCD display to ensure it showed clear and accurate messages during different system operations. The main goal was to confirm that the display provided the correct feedback to the user in real-time.

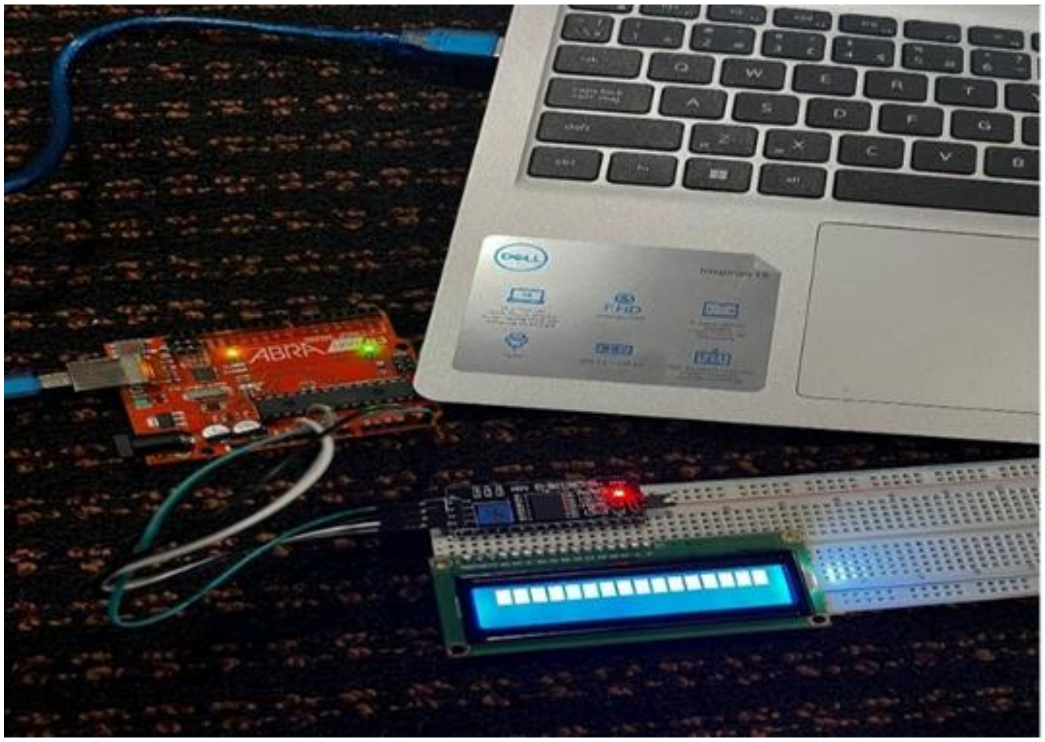
What We Planned: We wanted the LCD to display helpful messages like “Motion Detected,” “Enter Password,” “Access Granted,” or “Wrong Password” at the right time. It was important to check if the text was readable, updated quickly, and reflected the system’s status accurately.

What We Thought: We expected the LCD to immediately show a message when motion was detected and ask for a password. After the correct password was entered, it would display “Access Granted” and turn off the alarm. In case of a wrong password, it would display “Wrong Password” and keep the alarm active.

What We Used

- **16x2 I2C LCD Display** – For displaying system messages
- **Arduino Uno R3** – To send messages to the LCD
- **Buzzer and Keypad** – For input and alerts
- **Laptop with Arduino IDE** – For coding and testing

What We Tested as shown in figure below:



- Message Accuracy Test – Checked if the correct messages appeared at the right time.
- Readability Test – Ensured the text was clear and easy to read.
- Real-Time Update Test – Verified the display updated without delay during events like motion detection or password input.
- Error Message Test – Confirmed that "Wrong Password" appeared for incorrect input.
- Password Confirmation Test – Made sure the display showed "Access Granted" when the correct password was entered.
- These tests confirmed that the LCD provided accurate, clear, and timely feedback to the user.

- ## Serial Communication

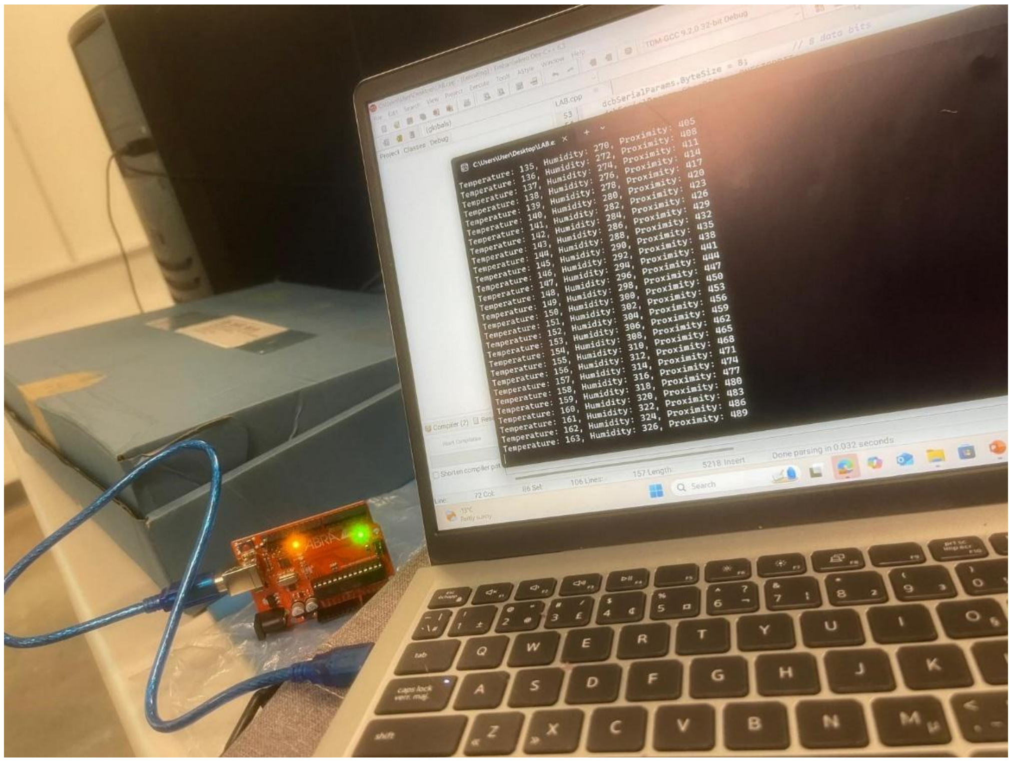
  We tested the serial communication between the Arduino and the computer to ensure accurate data transfer. The system was designed to send real-time data, including temperature, humidity, and proximity, along with motion detection alerts and password status.

What We Planned: Our goal was to confirm that the Arduino could send sensor data correctly through the serial port. We also checked if the computer displayed this data without any delays or errors. The messages on the screen had to match the actual sensor readings and system status.

What We Used:

- **Arduino Uno R3** – To collect and send sensor data
- **Temperature and Humidity Sensor (DHT11)** – For environmental readings
- **Ultrasonic Sensor (HC-SR04)** – To measure proximity
- **PIR Sensor** – For motion detection
- **Laptop with Arduino IDE** – To view data on the serial monitor
- **Keypad and Buzzer** – For input and alarms

What We Tested is shown in the figure below:



```
dcbSerialParams.ByteSize = 8;
```

```
Temperature: 135, Humidity: 270, Proximity: 405
Temperature: 136, Humidity: 272, Proximity: 408
Temperature: 137, Humidity: 274, Proximity: 411
Temperature: 138, Humidity: 276, Proximity: 414
Temperature: 139, Humidity: 278, Proximity: 417
Temperature: 140, Humidity: 280, Proximity: 420
Temperature: 141, Humidity: 282, Proximity: 423
Temperature: 142, Humidity: 284, Proximity: 426
Temperature: 143, Humidity: 286, Proximity: 429
Temperature: 144, Humidity: 288, Proximity: 432
Temperature: 145, Humidity: 290, Proximity: 435
Temperature: 146, Humidity: 292, Proximity: 438
Temperature: 147, Humidity: 294, Proximity: 441
Temperature: 148, Humidity: 296, Proximity: 444
Temperature: 149, Humidity: 298, Proximity: 447
Temperature: 150, Humidity: 300, Proximity: 450
Temperature: 151, Humidity: 302, Proximity: 453
Temperature: 152, Humidity: 304, Proximity: 456
Temperature: 153, Humidity: 306, Proximity: 459
Temperature: 154, Humidity: 308, Proximity: 462
Temperature: 155, Humidity: 310, Proximity: 465
Temperature: 156, Humidity: 312, Proximity: 468
Temperature: 157, Humidity: 314, Proximity: 471
Temperature: 158, Humidity: 316, Proximity: 474
Temperature: 159, Humidity: 318, Proximity: 477
Temperature: 160, Humidity: 320, Proximity: 480
Temperature: 161, Humidity: 322, Proximity: 483
Temperature: 162, Humidity: 324, Proximity: 486
Temperature: 163, Humidity: 326, Proximity: 489
```

- Sensor Data Test – Verified that temperature, humidity, and proximity data were sent and displayed correctly.
- Motion Detection Test – Checked if the system sent "Motion Detected" when motion occurred.
- Password Entry Test – Ensured the correct or wrong password message was sent based on user input.
- Data Accuracy Test – Compare the sensor readings with actual environmental values.
- Real-Time Response Test – Measured how quickly the serial monitor updated data after detection.
- The results showed that the system effectively communicated with the computer, providing real-time updates and ensuring reliable monitoring.

10.3 Testing Phase 1

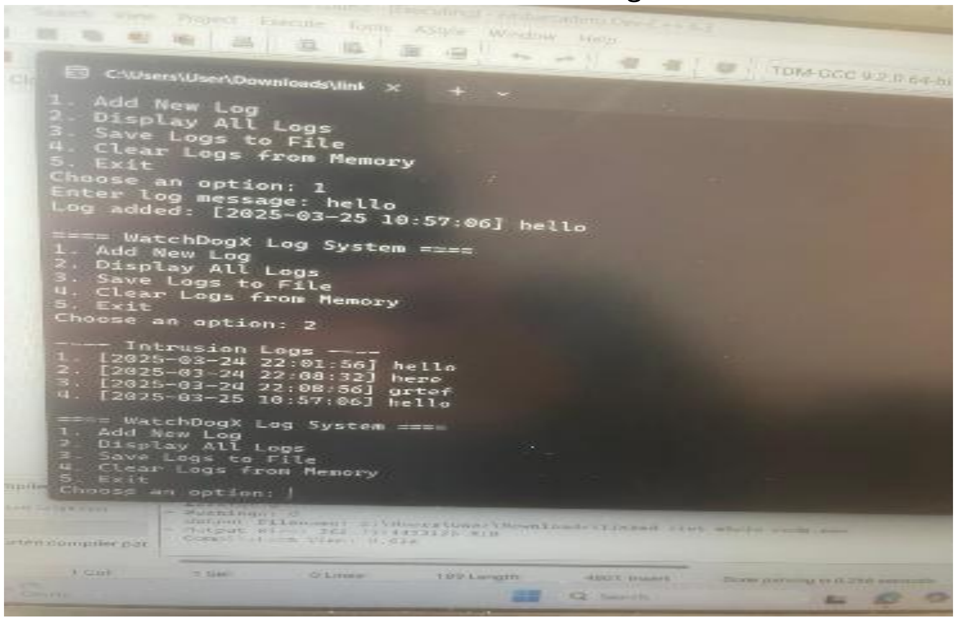
- **Linked List**

What We Planned: We aimed to check if the linked list could store multiple intrusion events, assign proper timestamps, and display the logs in chronological order. Additionally, we verified if the system could handle log additions, deletions, and updates without errors.

What We Used

- **Arduino Uno R3** – To send motion detection data
- **Laptop with Custom Software** – To manage logs using linked lists
- **Linked List Data Structure** – For storing and organizing intrusion events
- **Keypad and Buzzer** – To simulate intrusion scenarios

What We Tested is shown in the figure below:



```
C:\Users\User\Downloads\lin
1. Add New Log
2. Display All Logs
3. Save Logs to File
4. Clear Logs from Memory
5. Exit
Choose an option: 1
Enter log message: hello
Log added: [2025-03-25 10:57:06] hello

==== WatchDogX Log System ====
1. Add New Log
2. Display All Logs
3. Save Logs to File
4. Clear Logs from Memory
5. Exit
Choose an option: 2

---- Intrusion Logs ----
1. [2025-03-24 22:01:56] hello
2. [2025-03-24 22:08:32] here
3. [2025-03-24 22:08:56] grtef
4. [2025-03-25 10:57:06] hello

==== WatchDogX Log System ====
1. Add New Log
2. Display All Logs
3. Save Logs to File
4. Clear Logs from Memory
5. Exit
Choose an option:
```

- Event Logging Test – Checked if each detected motion was added as a new node in the linked list with a correct timestamp.
- Order Verification Test – Ensured the events were stored in the correct chronological order.
- Log Display Test – Verified if the software displayed logs accurately from the linked list.
- Deletion Test – Confirmed the system could remove specific logs without affecting the remaining data.
- Performance Test – Monitored the system's response time as the number of logs increased.
- The results showed that the linked list efficiently handled and organized events, providing accurate timestamps and maintaining the correct order of events.

• File Storage Testing

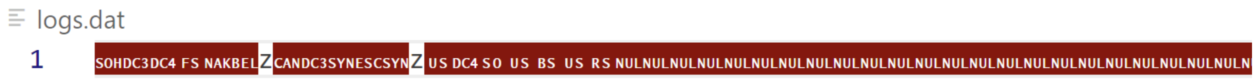
We tested the file storage system of WatchDogX to ensure that intrusion logs were saved securely and could be accessed, updated, or deleted when needed. The primary goal was to confirm that the logs were stored in an organized manner with correct timestamps and that the data remained intact during file operations.

What We Planned: We wanted to verify that the system could create, read, and update log files without errors. It was also important to check if the files were encrypted for security purposes and that log data could be restored after restarting the application.

What We Used

- **Laptop with Custom Software** – For file management
- **Linked List Data Structure** – For organizing logs before storage
- **File System** – For storing data in encrypted log files
- **Password Authentication** – To ensure only authorized users can access the files

What We Tested is shown in the figure below:



logs.dat

```
SOHDC3DC4 FS NAKBELZCANDC3SYNESCSYNZUS DC4 SO US BS US RS NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULN
```

File Creation Test – Checked if a log file was created when the first intrusion event was detected.

- Data Writing Test – Verified that the correct event data, including timestamps, was saved to the file.
- Data Reading Test – Ensured the software could read and display log data from the file.
- File Encryption Test – Confirmed that the logs were stored securely using XOR encryption.
- Data Persistence Test – Restarted the application to ensure that previous logs were retained and accessible.
- File Deletion Test – Checked if log files could be deleted by authorized users without errors.
- The results showed that the file storage system operated reliably, maintaining data accuracy and security while allowing authorized access for log management.

• User interface (UI) testing

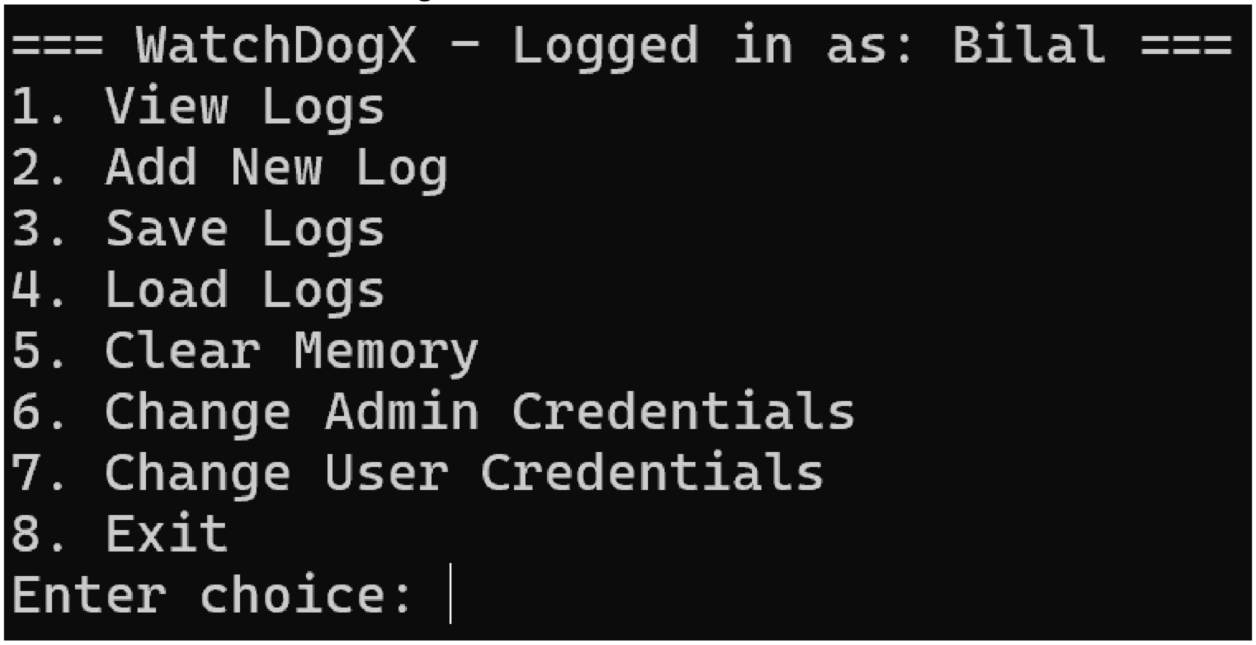
We tested the user interface (UI) of the WatchDogX system to ensure it was easy to navigate and displayed information accurately. The primary goal was to confirm that users could interact with the system effectively using the available options for viewing, adding, or deleting logs, as well as accessing and managing data securely.

What We Planned: We aimed to check if the UI provided clear instructions, responded correctly to user inputs, and displayed data without errors. The focus was on verifying the accuracy of log displays, the functionality of buttons and menus, and the security of password-protected sections.

What We Used

- **Laptop with Custom Software** – To run and test the user interface
- **Linked List Data Structure** – For managing and displaying logs
- **Password Authentication System** – For secure access
- **Serial Communication** – For receiving real-time data from Arduino

What We Tested as shown in figure below:



```
=== WatchDogX - Logged in as: Bilal ===
1. View Logs
2. Add New Log
3. Save Logs
4. Load Logs
5. Clear Memory
6. Change Admin Credentials
7. Change User Credentials
8. Exit
Enter choice:
```

1. Navigation Test – Checked if users could move through the menu options without difficulty.
2. Data Display Test – Verified that intrusion logs, timestamps, and system messages were displayed correctly.

3. Password Protection Test – Ensured only authorized users could access sensitive data or delete logs.
4. Functionality Test – Confirmed that actions like viewing, adding, and deleting logs worked without errors.
5. Error Handling Test – Tested how the UI responded to invalid inputs or failed operations.

10.2 Test Results

Test Summary Table

Test name	Input	Expected output	Actual output	Comments	GitHub link
PIR Sensor Test	Motion within sensor range	Motion Detected on LCD, Buzzer sounds	As expected.	Sensor-detected motion reliably	GitHub-PIR Test Code
No Motion Test	No movement in the sensor range	No alert	As expected.	No false detections were recorded.	GitHub-PIR Test code
Password entry test	The correct password was entered on the keypad	Access granted, on LCD, buzzer turned off	As expected.	The system responds correctly to the correct password	GitHub-Keypad test code
Wrong password test	The incorrect password entered on the keypad	Wrong password on LCD, buzzer remains on	As expected,	The alarm remains active for wrong inputs	GitHub-keypad test code
LCD display test	Various system events	Relevant messages displayed on LCD	As expected.	The display provided accurate feedback	GitHub-LCD test code

Serial communication test	Sensor data, status updates	Data is transmitted to the serial monitor.	As expected	Smooth communication between Arduino and laptop	GitHub- serial test code
Linked list logging test	Motion detected, events added	Logs stored with the correct timestamp	As expected,	Events are stored in the correct order using a linked list	GitHub- linked list test code
File storage test	Logs stored file	Data saved in encrypted format	As expected.	Logs were encrypted and stored securely	GitHub- file storage test code.

GitHub Link for all the testing codes: https://github.com/Bilal2866/PROJECT-2/blob/Week-5/Testing%20Codes.zip

10.3 Observations

- The system detected motion accurately with minimal false positives.
- Password input and verification worked effectively, ensuring secure access.
- The LCD displayed clear and timely messages for all system events.
- Serial communication was stable, providing real-time updates.
- Linked list logging maintained data efficiently and in the correct order.
- File storage was reliable, with secure encryption and successful data retrieval.
- Overall, the system performed well, meeting its functional and security goals.

11. Discussion

11.1 Analysis of Results

- The WatchDogX project successfully met its initial objectives of creating a secure, role-based intrusion logging system with both user and admin access levels. The implementation of XOR-based encryption ensured the basic protection of credentials and log data, while the modular logger library allowed a clean separation of log management features from the main application logic.
- In our final demo, we showcased a fully integrated Arduino and desktop software system. The hardware operated stably, and real-time sensor data was successfully transmitted via serial communication to the desktop app.
- The desktop interface featured secure login for both Admin and User roles, with password protection and XOR encryption for credentials and logs. A linked list was used to manage log data, which could be saved and loaded from encrypted binary files.
- We also explained the system's architecture and walked through key features clearly. While most features worked as expected, the Arduino reboot functionality was not operational during the demo and will be completed later.

11.2 Limitations

While WatchDogX works well, there are a few limitations we noticed during testing.

1. **Sensor sensitivity**

The PIR sensor sometimes gave false alerts if there were quick temperature changes, shadows, or small movements (like pets) this could make the alarm go off even when there's no real threat.

2. **Limited range and angle**

The motion sensor only works within a certain distance and angle. If someone moves outside this range, it might not detect them.

3. **Wired connection**

The system depends on a USB cable to send data from the Arduino to the laptop. This limits how far the hardware can be placed from the computer.

4. **Basic User Interface**

The current interface is command- line based, which is simple but not user friendly for everyone. Some users may find it hard to use or understand.

5. **Power dependency**

The system needs to be plugged in. If there is power cut, it will stop working unless a backup power system is added.

11.3 Potential Improvements

While WatchDogX works well in its current form, there are several ways it can be improved in the future.

1. **Wireless communication.**
   Instead of using a USB cable, we could use Wi-Fi or Bluetooth to send data wirelessly from the Arduino to a computer or phone.

2. **Mobile app support**
   A simple mobile app could help users get alerts and check logs from anywhere making the system more flexible.

3. **More sensors**
   Adding more sensors like temperature, door contact, or ultrasonic distance sensors would help cover more types of intrusions.

4. **Better User Interface**
   We currently use a command line interface. Replacing it with a simple menu-based window would make the system easier to use.

5. **Log backup options**
   We can add features to save logs in files like PDF or Excel and allow users to back them up automatically.

6. **Battery support**
   Adding a battery backup will help the system keep working even if the power goes out.

7. **Smarter Motion Detection**
   Later versions could use basic AI to avoid false alarms caused by pets or small movements.

8. **User Interface**
   Wrong logs can be highlighted for quick understanding and detection.

12. Conclusion & Future Work

In this project, we built a security system called WatchDogX. It can detect motion, make a sound using a buzzer, and see messages on an LCD screen. The alarm can be turned off by typing a password. It can also save all events (like intrusions) into a file so we can check them later.

We used an Arduino, a PIR motion sensor, an LCD, a keypad, and some simple software to build and test the system. We also used a password system and used encryption to protect the data.

What we learned:

- How to connect and use sensors with Arduino.
- How to write and upload code.
- How to show messages on an LCD screen.
- How to save logs using linked lists and files.
- How to make a basic software interface and protect it with a password.

Future improvements:

In the future, we can make the system better by

- Making it work wirelessly (no USB cable needed.)
- Creating a mobile app to get alerts on phones.
- Adding more sensors like door or temperature sensors.
- Improving the software design with better graphics.
- Using stronger security to protect the data.

This project was a great learning experience and can be used in real life to protect homes, shops, or storage rooms from intruders.

13. References

[1] Arduino: Electronics.com.bd. (n.d.). *Arduino Uno R3 SMD Atmega328 IC*. Retrieved March 25, 2025, from https://www.electronics.com.bd/Arduino-Uno-R3-SMD-Atmega328-IC-Arduino-Uno-Arduino-Uno-R3-R3-Original-ATmega328P-CH340-SMD-SMD-IC-Price-in-online-BD-Bangladesh?route=product/product

[2] Breadboard: Canada Robotix. (n.d.). *Arduino Uno R3 Compatible – CH340*. Retrieved March 25, 2025, from https://www.canadarobotix.com/products/160

[3] LCD: PiShop.ca. (n.d.). *1602 LCD Display Module DC 5V 16x2 (Blue Backlight)*. Retrieved March 25, 2025, from https://www.pishop.ca/product/1602-lcd-display-module-dc-5v-16x2-blue-blacklight/

[4] Keypad: ElectroPeak. (n.d.). *Flat Keypad 4x4 Matrix – Membrane Type*. Retrieved March 25, 2025, from https://electropeak.com/flat-keypad-4x4

[5] Buzzer: PCBoard.ca. (n.d.). *Mini Piezo Buzzer*. Retrieved March 25, 2025, from https://www.pcboard.ca/minipiezo-buzzer

[6] Jumper Wires: Canada Robotix. (n.d.). *Premium Jumper Wires 20 cm (40-pack)*. Retrieved March 25, 2025 from https://www.canadarobotix.com/products/2293

[7] Laptop: HP Canada. (n.d.). *HP Laptop – Model 7Q024UA#ABL*. Retrieved March 25, 2025, from https://www.hp.com/ca-en/shop/product.aspx?id=7q024ua&opt=abl&sel=ntb

[8] Figure 1: Autodesk. (2025). *Tinkercad Circuits – Motion detection with Arduino*. Tinkercad. Retrieved from https://www.tinkercad.com

[9] Figure 2: Autodesk. (2025). *Tinkercad Circuits – Motion detection with Arduino*. Tinkercad. Retrieved from https://www.tinkercad.com

[10] Arduino. (n.d.). *Arduino Uno Rev3*. Arduino Documentation. Retrieved March 27, 2025, from https://docs.arduino.cc/hardware/uno-rev3

[11] ELEGOO. (n.d.). *Upgraded 37-in-1 Sensor Modules Kit with Tutorial Compatible with Arduino IDE UNO R3 MEGA Nano*. Retrieved March 27, 2025, from https://www.amazon.com/ELEGOO-Upgraded-Tutorial-Compatible-MEGA2560/dp/B01MG49ZQ5

[12] Testing Codes: https://github.com/Bilal2866/PROJECT-2/blob/Week-5/Testing%20Codes.zip

[13] Arduino Forum. (2020). *Motion sensor with noise and LED for beginners*. Retrieved March 27, 2025, from https://forum.arduino.cc/t/motion-sensor-with-noise-and-led-for-beginners/685127

[14] ArduinoIntro. (n.d.). *DIY Motion Detection Alarm System Using PIR Sensor, I2C LCD and Piezo Buzzer*. Retrieved March 27, 2025, from https://arduinointro.com/articles/projects/diy-motion-detection-alarm-system-using-pir-sensor-i2c-lcd-and-piezo-buzzer

[15] Tinkercad. (n.d.). *PIR sensor and buzzer circuit with Arduino*. Autodesk. Retrieved March 27, 2025, from https://www.tinkercad.com/things/0mzV5mFKYrC-pir-sensor-and-buzzer-circuit-with-arduino

[16] Green Economy Canada. (n.d.). *Conestoga College*. https://greeneconomy.ca/business/conestoga-college/

[17] Diagrams.net. (n.d.). *diagrams.net – Free Online Diagram Software*. Retrieved April 8, 2025, from https://www.diagrams.net

14. Appendices

14.1 Full Code Listing

Header file/ Client File

```
#ifndef WATCHDOGX_H
#define WATCHDOGX_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>

// XOR encryption key for log obfuscation
#define ENCRYPTION_KEY 0xAB

// File to store encrypted log data
#define LOG_FILE "logs.dat"

// ========================
// Log Data Structure
// ========================
typedef struct Log {
   char message[128];     // Log message content
   char timestamp[32];    // Timestamp of the log
   struct Log* next;      // Pointer to the next log node (linked list)
} Log;

// ========================
// Core Log Operations
// ========================

// Display all logs from memory (linked list)
void displayLogs();

// Add a new log entry (manual input)
void addLog();

// Clear all logs from memory and file
void clearLogs();

// Save current logs to binary file with encryption
void saveLogsToFile();

// Load logs from binary file into memory
void loadLogsFromFile();

// Add log directly from Arduino serial message
```

```
void addLogFromSerial(const char* message);

// ========================
// Utility & Credential Functions
// ========================

// Encrypt/Decrypt data using XOR algorithm
void xorEncryptDecrypt(unsigned char* data, size_t len);

// Get current system time as a formatted string
void getTimestamp(char* buffer, int size);

// Display ASCII welcome banner
void showWelcome();

// ========================
// Arduino Serial Communication
// ========================

// Auto-detect the COM port used by Arduino
char* detectArduinoPort();

// Open serial communication with Arduino
HANDLE openSerialPort(const char* portName);

// Read one line from the serial port
int readLineFromSerial(HANDLE hSerial, char* buffer, int maxLen);

// Continuously display live log feed from Arduino
void liveLogFeedFromArduino();

// Reboot Arduino by toggling DTR
void rebootArduino(const char* portName);  // NEW: Added reboot support

#endif
```

Implementation file/ .C File

```c
#include "WatchDogX.h"
#include <windows.h>
#include <setupapi.h>
#include <stdio.h>
#include <string.h>
#include <initguid.h>
#include <devguid.h>
#include <conio.h>

// Global pointer to the head of the log linked list
Log* head = NULL;

// Encrypts or decrypts data using XOR with a fixed key
void xorEncryptDecrypt(unsigned char* data, size_t len) {
   for (size_t i = 0; i < len; i++)
      data[i] ^= ENCRYPTION_KEY;
}

// Fills the provided buffer with the current system timestamp
void getTimestamp(char* buffer, int size) {
   time_t now = time(NULL);
   struct tm* t = localtime(&now);
   strftime(buffer, size, "%Y-%m-%d %H:%M:%S", t);
}

// Pauses the screen until the user presses Enter
void pauseScreen() {
   printf("\nPress Enter to return to menu...");
   getchar();
}

// Clears all logs from memory by freeing the linked list
void clearLogs() {
   Log* temp = head;
   while (temp) {
      Log* next = temp->next;
      free(temp);
      temp = next;
   }
   head = NULL;
   printf("Logs cleared from memory.\n");
   Sleep(1000);
}

// Displays all logs currently in memory
void displayLogs() {
   if (!head) {
      printf("No logs to display.\n");
```

```
        pauseScreen();
        return;
    }

    int count = 1;
    Log* temp = head;
    while (temp) {
        printf("%d. [%s] %s\n", count++, temp->timestamp, temp->message);
        temp = temp->next;
    }
    pauseScreen();
}

// Saves logs to a binary file with XOR encryption
void saveLogsToFile() {
    FILE* file = fopen(LOG_FILE, "wb");
    if (!file) {
        printf("Error saving logs.\n");
        return;
    }

    Log* temp = head;
    while (temp) {
        Log encrypted = *temp;
        xorEncryptDecrypt((unsigned char*)encrypted.message, strlen(encrypted.message));
        xorEncryptDecrypt((unsigned char*)encrypted.timestamp, strlen(encrypted.timestamp));
        fwrite(&encrypted, sizeof(Log), 1, file);
        temp = temp->next;
    }

    fclose(file);
    printf("Logs saved to binary file.\n");
    Sleep(1000);
}

// Loads logs from the binary file into memory, decrypting them
void loadLogsFromFile() {
    clearLogs();
    FILE* file = fopen(LOG_FILE, "rb");
    if (!file) {
        printf("No saved logs found.\n");
        return;
    }

    Log temp;
    while (fread(&temp, sizeof(Log), 1, file) == 1) {
        xorEncryptDecrypt((unsigned char*)temp.message, strlen(temp.message));
        xorEncryptDecrypt((unsigned char*)temp.timestamp, strlen(temp.timestamp));

        Log* newLog = malloc(sizeof(Log));
        if (!newLog) {
            printf("Memory allocation failed.\n");
            break;
        }

        *newLog = temp;
        newLog->next = NULL;

        if (!head) head = newLog;
        else {
            Log* curr = head;
            while (curr->next) curr = curr->next;
            curr->next = newLog;
        }
    }

    fclose(file);
    printf("Logs loaded from binary file.\n");
    Sleep(1000);
}
// Adds a manual log from user input
void addLog() {
    int type;
    char tag[10];
    char msg[100];

    printf("Select Log Type:\n1. INFO\n2. WARNING\n3. ALERT\nChoice: ");
    scanf("%d", &type);
    while (getchar() != '\n');  // Clear input buffer

    switch (type) {
        case 1: strcpy(tag, "[INFO]"); break;
        case 2: strcpy(tag, "[WARNING]"); break;
        case 3: strcpy(tag, "[ALERT]"); break;
        default: strcpy(tag, "[LOG]"); break;
    }

    printf("Enter log message: ");
    fgets(msg, sizeof(msg), stdin);
    msg[strcspn(msg, "\n")] = '\0';  // Remove trailing newline

    Log* newLog = malloc(sizeof(Log));
    if (!newLog) {
        printf("Memory allocation failed.\n");
        return;
    }

    snprintf(newLog->message, sizeof(newLog->message), "%s %s", tag, msg);
    getTimestamp(newLog->timestamp, sizeof(newLog->timestamp));
    newLog->next = NULL;
    if (!head) head = newLog;
    else {
        Log* temp = head;
        while (temp->next) temp = temp->next;
        temp->next = newLog;
    }

    saveLogsToFile();  // Auto-save
    printf("Log added and saved.\n");
    Sleep(1000);
}

// Adds a log entry from an incoming serial message
void addLogFromSerial(const char* message) {
    Log* newLog = malloc(sizeof(Log));
    if (!newLog) {
        printf("Memory allocation failed.\n");
        return;
    }

    snprintf(newLog->message, sizeof(newLog->message), "[ARDUINO] %s", message);
    getTimestamp(newLog->timestamp, sizeof(newLog->timestamp));
    newLog->next = NULL;

    if (!head) head = newLog;
    else {
        Log* temp = head;
        while (temp->next) temp = temp->next;
        temp->next = newLog;
    }

    saveLogsToFile();  // Auto-save on message receive
}

// Open and configure serial port
HANDLE openSerialPort(const char* portName) {
    HANDLE hSerial = CreateFileA(portName, GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
    if (hSerial == INVALID_HANDLE_VALUE) return INVALID_HANDLE_VALUE;

    DCB dcbSerialParams = {0};
    dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
    GetCommState(hSerial, &dcbSerialParams);

    dcbSerialParams.BaudRate = CBR_9600;
    dcbSerialParams.ByteSize = 8;
    dcbSerialParams.StopBits = ONESTOPBIT;
    dcbSerialParams.Parity   = NOPARITY;

    SetCommState(hSerial, &dcbSerialParams);
    COMMTIMEOUTS timeouts = {0};
    timeouts.ReadIntervalTimeout = 50;
    timeouts.ReadTotalTimeoutConstant = 50;
    timeouts.ReadTotalTimeoutMultiplier = 10;
    SetCommTimeouts(hSerial, &timeouts);

    return hSerial;
}

// Reads one line from serial port into buffer
int readLineFromSerial(HANDLE hSerial, char* buffer, int maxLen) {
    char c;
    DWORD bytesRead;
    int i = 0;

    while (i < maxLen - 1) {
        if (!ReadFile(hSerial, &c, 1, &bytesRead, NULL)) return -1;
        if (bytesRead == 0) continue;
        if (c == '\n') break;
        buffer[i++] = c;
    }
    buffer[i] = '\0';
    return i;
}

// Displays live serial logs until key is pressed
void liveLogFeedFromArduino() {
    char* port = "\\\\.\\COM3";  // Static COM port; can be updated to use detectArduinoPort()

    HANDLE serial = openSerialPort(port);
    if (serial == INVALID_HANDLE_VALUE) {
        printf("Failed to open %s.\n", port);
        Sleep(1500);
        return;
    }

    system("cls");
    printf("=== Live Log Feed (Press any key to exit) ===\n");

    char line[128];
    while (!_kbhit()) {
        int len = readLineFromSerial(serial, line, sizeof(line));
        if (len > 0) {
            printf("%s\n", line);
            addLogFromSerial(line);
        }
        Sleep(10);
    }
    _getch();
    CloseHandle(serial);
}

// Reboots Arduino by opening port at 1200 baud (triggers reset)
void rebootArduino(const char* portName) {
    HANDLE hSerial = CreateFileA(portName, GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0,
NULL);
    if (hSerial == INVALID_HANDLE_VALUE) {
        printf("Failed to open %s for reboot.\n", portName);
        Sleep(1500);
        return;
    }

    DCB dcbParams = {0};
    dcbParams.DCBlength = sizeof(dcbParams);
    GetCommState(hSerial, &dcbParams);
    dcbParams.BaudRate = 1200;  // Magic number for Arduino reset
    SetCommState(hSerial, &dcbParams);

    CloseHandle(hSerial);
    printf("Sent reboot signal to Arduino on %s\n", portName);
    Sleep(2000);  // Wait for reboot
}

// Shows the app welcome header
void showWelcome() {
    system("cls");
    printf("========================================\n");
    printf("        W A T C H D O G   X        \n");
    printf("========================================\n");
    printf("    Secure Intrusion Logger & Monitor   \n");
    printf("========================================\n\n");
}
```

Main.c Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include <conio.h>
#include "WatchDogX.h"

#define ADMIN_CREDENTIALS_FILE "admin_credentials.txt"
#define USER_CREDENTIALS_FILE  "user_credentials.txt"

int isAdmin = 0;            // Track current user role
char currentUser[50] = "";    // Store logged-in username

// Flush leftover input from stdin
void clearStdin() {
   int c;
   while ((c = getchar()) != '\n' && c != EOF);
}

// Take masked password input (***** style)
void getMaskedInput(char* buffer, int maxLen) {
   char ch;
   int i = 0;
   while ((ch = _getch()) != '\r' && i < maxLen - 1) {
      if (ch == '\b' && i > 0) {
         i--;
         printf("\b \b");
      } else if (ch != '\b') {
         buffer[i++] = ch;
         printf("*");
      }
   }
   buffer[i] = '\0';
   printf("\n");
}

// Create new credentials for admin/user and encrypt
void setCredentials(const char* filePath) {
   showWelcome();
   FILE* file = fopen(filePath, "wb");
   if (!file) {
      printf("Error creating credentials file.\n");
      return;
   }

   char username[50], password[50];
   printf("Set username: ");
   fgets(username, sizeof(username), stdin);
```

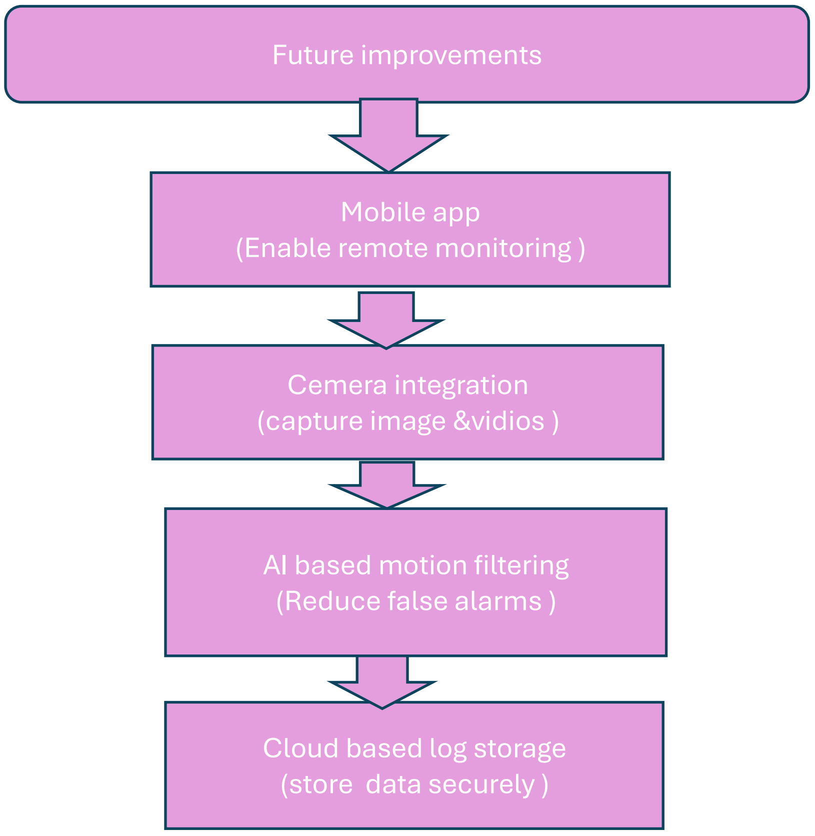
```
    username[strcspn(username, "\n")] = '\0';

    printf("Set password: ");
    getMaskedInput(password, sizeof(password));

    xorEncryptDecrypt((unsigned char*)username, strlen(username));
    xorEncryptDecrypt((unsigned char*)password, strlen(password));

    fprintf(file, "%s\n%s", username, password);
    fclose(file);
    printf("Credentials saved.\n");
    Sleep(1000);
}

// Verify login against saved encrypted credentials
int verifyCredentials(const char* filePath) {
    FILE* file = fopen(filePath, "rb");
    if (!file) {
        printf("No credentials found. Please set them.\n");
        setCredentials(filePath);
        return verifyCredentials(filePath);
    }

    char storedUser[50], storedPass[50];
    fgets(storedUser, sizeof(storedUser), file);
    fgets(storedPass, sizeof(storedPass), file);
    fclose(file);

    storedUser[strcspn(storedUser, "\n")] = '\0';
    storedPass[strcspn(storedPass, "\n")] = '\0';

    xorEncryptDecrypt((unsigned char*)storedUser, strlen(storedUser));
    xorEncryptDecrypt((unsigned char*)storedPass, strlen(storedPass));

    char inputUser[50], inputPass[50];
    printf("Enter username: ");
    fgets(inputUser, sizeof(inputUser), stdin);
    inputUser[strcspn(inputUser, "\n")] = '\0';

    printf("Enter password: ");
    getMaskedInput(inputPass, sizeof(inputPass));

    if (strcmp(inputUser, storedUser) == 0 && strcmp(inputPass, storedPass) == 0) {
        strcpy(currentUser, inputUser);
        return 1;
    }

    return 0;
}
```

```
// Allow user to choose admin/user role and login
void loginMenu() {
  showWelcome();
  int role;
  printf("Choose Role:\n1. Admin\n2. User\nEnter choice: ");
  scanf("%d", &role);
  clearStdin();

  if (role == 1) {
    if (verifyCredentials(ADMIN_CREDENTIALS_FILE)) isAdmin = 1;
    else exit(1);
  } else if (role == 2) {
    if (verifyCredentials(USER_CREDENTIALS_FILE)) isAdmin = 0;
    else exit(1);
  } else {
    printf("Invalid selection.\n");
    exit(1);
  }
}

// Main menu: Admin has extended options
void showMenu() {
  int choice;
  do {
    showWelcome();
    printf("=== Logged in as: %s (%s) ===\n", currentUser, isAdmin ? "Admin" : "User");
    printf("1. View Logs\n");
    if (isAdmin) {
      printf("2. Add New Log\n");
      printf("3. Save Logs\n");
      printf("4. Load Logs\n");
      printf("5. Clear Memory\n");
      printf("6. Change Admin Credentials\n");
      printf("7. Change User Credentials\n");
      printf("8. Exit\n");
      printf("9. Live Log Feed from Arduino\n");
      printf("10. Reboot Arduino\n");
    } else {
      printf("2. Load Logs\n");
      printf("3. Exit\n");
    }

    printf("Enter choice: ");
    scanf("%d", &choice);
    clearStdin();

    if (isAdmin) {
      switch (choice) {
        case 1: displayLogs(); break;
        case 2: addLog(); break;
```

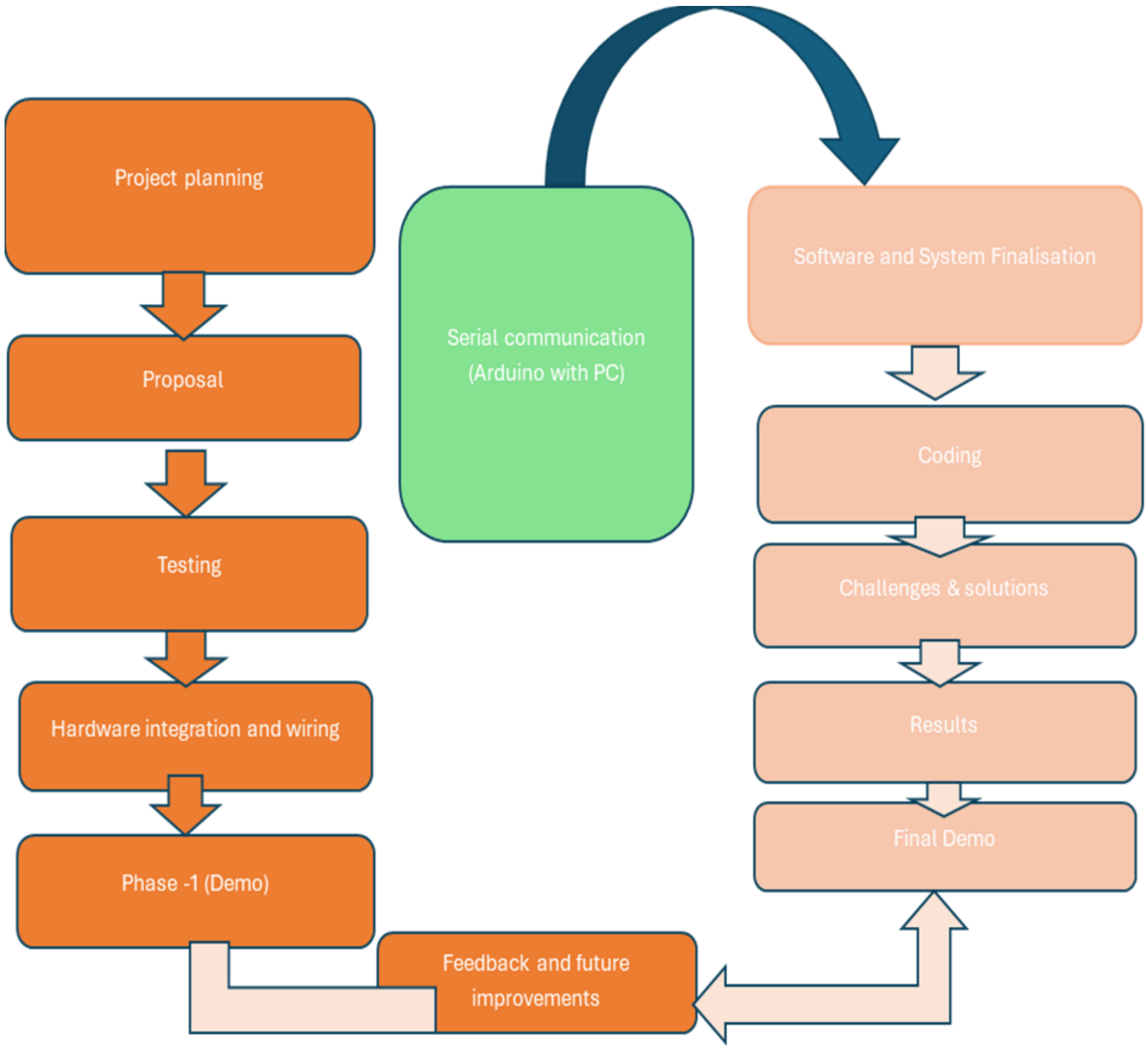
```
            case 3: saveLogsToFile(); break;
            case 4: loadLogsFromFile(); break;
            case 5: clearLogs(); break;
            case 6: setCredentials(ADMIN_CREDENTIALS_FILE); break;
            case 7: setCredentials(USER_CREDENTIALS_FILE); break;
            case 8: printf("Exiting...\n"); break;
            case 9: liveLogFeedFromArduino(); break;
            case 10: rebootArduino("\\\\.\\COM3"); break; // Use detectArduinoPort() optionally
            default: printf("Invalid input!\n"); Sleep(1000);
         }
      } else {
         switch (choice) {
            case 1: displayLogs(); break;
            case 2: loadLogsFromFile(); break;
            case 3: printf("Exiting...\n"); break;
            default: printf("Invalid input!\n"); Sleep(1000);
         }
      }
   } while ((isAdmin && choice != 8) || (!isAdmin && choice != 3));
}

// Main function
int main() {
   showWelcome();
   loginMenu();
   loadLogsFromFile();    // Pre-load logs into memory
   showMenu();          // Show options
   saveLogsToFile();      // Save before exiting
   clearLogs();          // Free memory
   return 0;
}
```

14.2 Additional Diagrams or Tables

The Diagram below shows what improvements can be made to WatchDogX:



This Diagram Illustrates the whole planning and implementation process:


Project planning
Proposal
Testing
Hardware integration and wiring
Phase -1 (Demo)
Feedback and future improvements
Serial communication (Arduino with PC)
Software and System Finalisation
Coding
Challenges & solutions
Results
Final Demo