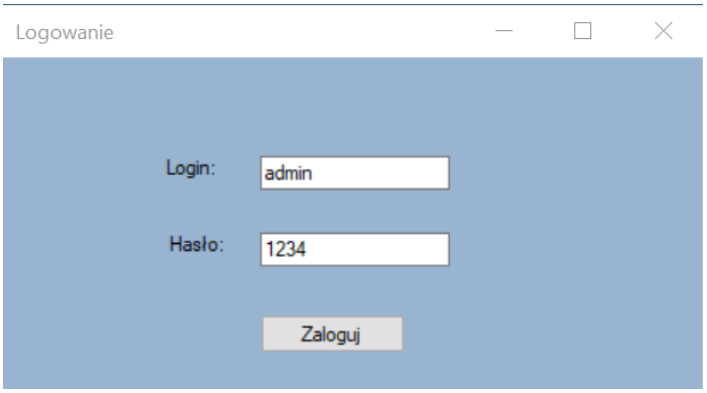
Aleksandra Janczewska 175962

Architektura Systemów Komputerowych Laboratorium

Sprawozdanie

Zadanie 6

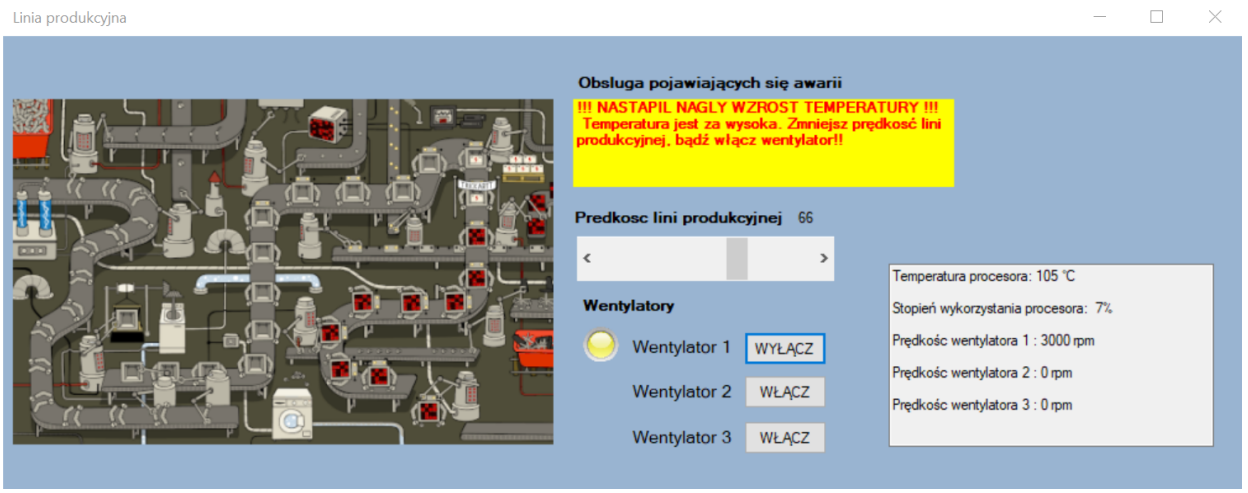
Celem tego zadania było stworzenie aplikacji, będącej symulatorem stanowiska dyspozytorskiego „linii produkcyjnej”. Zgodnie z wytycznymi stworzyłam aplikację, która swoje działanie rozpoczyna od widoku okna do logowania.



Listę zawierającą loginy i hasła, umożliwiające dostęp do aplikacji, stworzyłam posługując się bazą danych.

```
LogindbEntities context = new LogindbEntities();
if (login_textBox.Text != string.Empty || password_textBox.Text != string.Empty)
{
    var user = context.AdminLogins.Where(a => a.UserName.Equals(login_textBox.Text)).FirstOrDefault();
    if(user != null)
    {
        if (user.Password.Equals(password_textBox.Text))
        {
            this.Hide();
            Form1 form = new Form1();
            form.Show();
        }
        else
        {
            MessageBox.Show("Podany login i/lub hasło są nieprawidłowe.");
        }
```

Po poprawnym zalogowaniu do aplikacji, otwiera się okno główne zawierające symulator linii produkcyjnej.



Wykorzystywane są tutaj informacje na temat parametrów pracy komputera, dokładniej jest to temperatura rdzenia procesora i stopień wykorzystania procesora.

```
private void CPU_temp()
{
    ManagementObjectSearcher searcher = new ManagementObjectSearcher(@"root\WMI", "SELECT * FROM MSAcpi_ThermalZoneTemperature");

    foreach (ManagementObject obj in searcher.Get())
    {
        temperature = Convert.ToDouble(obj["CurrentTemperature"].ToString());
        temperature = (temperature - 2732) / 10.0; // Convert the value to celsius degrees
    }
}
1 odwołanie
private void CPU_use()
{
    ManagementObjectSearcher searcher = new ManagementObjectSearcher("select * from Win32_PerfFormattedData_PerfOS_Processor");

    foreach (ManagementObject obj in searcher.Get())
    {
        usage = Convert.ToDouble(obj["PercentProcessorTime"].ToString());
    }
```

W symulatorze wpływ na temperaturę procesora wywołuje praca wentylatorów oraz prędkość linii produkcyjnej.

```
1 odwołanie
private void fanWorkd()
{
    if (fansParams.fan1_on)
    {
        temperature = temperature - 1;
    }
    if (fansParams.fan2_on)
    {
        temperature = temperature - 3;
    }
    if (fansParams.fan3_on)
    {
        temperature = temperature - 5;
    }
}
```

```
1 odwołanie
private void LineWork()
{
    if (fansParams.line_speed == 0) temperature -= 1;
    if (fansParams.line_speed > 0 && fansParams.line_speed <= 20) temperature += 1;
    if (fansParams.line_speed > 20 && fansParams.line_speed <= 40) temperature += 2;
    if (fansParams.line_speed > 40 && fansParams.line_speed <= 60) temperature += 3;
    if (fansParams.line_speed > 60 && fansParams.line_speed <= 80) temperature += 4;
    if (fansParams.line_speed > 80 && fansParams.line_speed <= 100) temperature += 5;

}
```

Manipulując tymi zmiennymi staramy się zapewnić dobre zachowanie linii produkcji. Działanie symulatora opiera się na pracy timerów. Pierwszy z nich zajmuje się ustalaniem zmian zachodzących w symulacji i wyświetlaniem informacji. Drugi odpowiada za sprawdzanie przytomności operator, co 30s wyświetlane jest okno, w którym trzeba potwierdzić obecność w ciagu 10s (w trakcie włączany jest jeszcze dźwięk alarmu), jeżeli nie zostanie ono wykonane, nastąpi wylogowanie z okna symulatora i otworzy się okno do logowania. Trzeci timer odpowiada za występowanie losowej awarii, korzystając z generatora liczb losowych. Losowa awaria uaktywnia się co 45s, posiada opcje: nagłej zmiany temperatury, nagłej zmiany prędkości linii produkcyjnej i awarii jednego z wentylatorów, uniemożliwiającej korzystanie z niego przez kolejne 20s.

```
int awaria = rand.Next(6);

switch (awaria)
{
    case 0:
        temperature = 100;
        awariaTemp = true;
        break;
    case 1:
        fansParams.line_speed = hScrollBar1.Value = 0;
        predkosc_label.Text = Convert.ToString(fansParams.line_speed);
        awariaLine0 = true;
        break;
    case 2:
        fansParams.line_speed = hScrollBar1.Value = 90;
        predkosc_label.Text = Convert.ToString(fansParams.line_speed);
        awariaLine1 = true;
        break;
    case 3:
        if (fansParams.fan1_on)
        {
            fan1_button.PerformClick();
            fan1_button.Enabled = false;
            czas_awarii = 0;
            awariaFan = true;
        }
        break;
    case 4:
```

Ostatni timer odpowiada za wyświetlanie obrazów linii produkcyjnej, w taki sposób by sprawiała wrażenie poruszającej się. Czas jego interwału jest ustalany na podstawie wyznaczonej, za pomocą suwaka, prędkości linii produkcyjnej.

```
private void timer4_Tick(object sender, EventArgs e)
{
    if (fansParams.line_speed == 0)
    {
        pictureBox1.Image = lista[indeks];
    }
    else
    {
        timer4.Interval = 1000/fansParams.line_speed ;
        indeks++;
        pictureBox1.Image = lista[indeks];
        if (indeks == 14) indeks = 0;
    }
```

Dyskusja osiągniętych wyników z wskazaniem wad i zalet napisanej aplikacji:

Jestem bardzo zadowolona z animacji linii produkcyjnej, przy pierwszym podejściu chciałam wykorzystać już gotową animację na podstawie plików gif o różnych prędkościach, niestety nie sprawdziło się to w tej aplikacji, dlatego też ostatecznie wybrałam metodę zmieniania obrazów na kolejne ujęcia. Podczas wykonywania tego projektu nauczyłam się tworzyć i wykorzystywać bazy danych oraz odczytywać w programie informacje takie jak parapety komputera PC. Nie udało mi się odczytać prędkości obrotowych wentylatorów mojego komputera, ich wartości w symulatorze zostały przypisane ręcznie. Z pewnością mogłabym się postarać o bardziej adekwatne opisy postępowań po wystąpieniu awarii.