ДОДАТОК А
Код авторизації користувачів

```
//Autorization
private async Task LoginAsync()
{
    var user = await _context.UsersChart.FirstOrDefaultAsync(u => u.Login == Login &&
u.Password == Password);
    if (user != null)
    {
        StatusBar = "Login successful!";
        IsAWr0Visible = false;
        IsAWr1Visible = false;
    }
    else
    {
        StatusBar = "Invalid login or password.";
    }
}
```

ДОДАТОК Б
Код оновлення графіків складу

```
// Chart
public async void UpdateMainChart()
{
    await LoadDataFromDatabaseAsync();
    await FirstRackFilterData();
    await SecondRackFilterData();
    await ThirdRackFilterData();
    await FourthRackFilterData();
    UpdateOrderCollection();
    UpdateProductionCollection();
}
public async Task LoadDataFromDatabaseAsync()
{
    var main = await GetInfoFromDatabaseAsync();
    await Dispatcher.UIThread.InvokeAsync(() =>
    {
        Main.Clear();
        Main.AddRange(main);
    });
}
public async Task<List<MainChart>> GetInfoFromDatabaseAsync()
{
    using var context = new AppDBContext();
    var data = await context.MainChart.ToListAsync();
    return data;
}
public async Task AddNewMainChartAsync(string name, string type)
{
    var newMainChart = new MainChart
    {
        Data = DateTime.Now,
        Name = name,
        Quantity = 0,
        MaterialType = type,
    };
    using var context = new AppDBContext();
    context.MainChart.Add(newMainChart);
    await context.SaveChangesAsync();
    await Dispatcher.UIThread.InvokeAsync(() =>
    {
        Main.Add(newMainChart);
    });
}
// First Rack Chart
private async Task FirstRackFilterData()
{
    var filteredItems = Main.Where(item => item.MaterialType == "Plate").ToList();
    foreach (var item in filteredItems)
    {
        item.Rack = 1;
    }
    FirstFilteredMain = new ObservableCollection<MainChart>(filteredItems);
    await UpdateDatabaseRackValues(filteredItems);
}
// Second Rack Chart
private async Task SecondRackFilterData()
{
```

```
    var filteredItems = Main.Where(item => item.MaterialType == "Wire").ToList();
    foreach (var item in filteredItems)
    {
        item.Rack = 2;
    }
    SecondFilteredMain = new ObservableCollection<MainChart>(filteredItems);
    await UpdateDatabaseRackValues(filteredItems);
}
// Third Rack Chart
private async Task ThirdRackFilterData()
{
    var filteredItems = Main.Where(item => item.MaterialType == "Pipe").ToList();
    foreach (var item in filteredItems)
    {
        item.Rack = 3;
    }
    ThirdFilteredMain = new ObservableCollection<MainChart>(filteredItems);
    await UpdateDatabaseRackValues(filteredItems);
}
// Fourth Rack Chart
private async Task FourthRackFilterData()
{
    var filteredItems = Main.Where(item => item.MaterialType == "Bag").ToList();
    foreach (var item in filteredItems)
    {
        item.Rack = 4;
    }
    FourthFilteredMain = new ObservableCollection<MainChart>(filteredItems);
    await UpdateDatabaseRackValues(filteredItems);
}
// Update database with new Rack values
private async Task UpdateDatabaseRackValues(IEnumerable<MainChart> items)
{
    using var context = new AppDBContext();
    foreach (var item in items)
    {
        var dbItem = await context.MainChart.FirstOrDefaultAsync(x => x.Id ==
item.Id);
        if (dbItem != null)
        {
            dbItem.Rack = item.Rack;
        }
    }
    await context.SaveChangesAsync();
}
```

ДОДАТОК В
Код керування зовнішніми замовленнями

```
// OrderChart
private async Task AddOrderAsync()
{
    using var context = new AppDBContext();
    FirstRackLinesVisibleForOrderComplete = false;
    SecondRackLinesVisibleForOrderComplete = false;
    ThirdRackLinesVisibleForOrderComplete = false;
    FourthRackLinesVisibleForOrderComplete = false;
    var mainChart = context.MainChart.FirstOrDefault(m => m.Name == OName);
    if (mainChart == null)
    {
        await Dispatcher.UIThread.InvokeAsync(() =>
        {
            StatusBar = "Name not found in main chart.";
        });
        return;
    }
    string orderStatus;
    if (mainChart.Quantity < OQuantity)
    {
        orderStatus = "In process";
        await Dispatcher.UIThread.InvokeAsync(() =>
        {
            StatusBar = "Insufficient products in stock.";
        });
    }
    else
    {
        mainChart.Quantity -= OQuantity;
        orderStatus = "Complete";
        await Dispatcher.UIThread.InvokeAsync(() =>
        {
            StatusBar = "Order successfully added.";
        });
    }
    var newOrder = new OrderChart
    {
        Data = DateTime.Now,
        Name = OName,
        MaterialType = mainChart.MaterialType,
        Quantity = OQuantity,
        Status = orderStatus
    };

    context.OrderChart.Add(newOrder);
    await context.SaveChangesAsync();
    await Dispatcher.UIThread.InvokeAsync(() =>
    {
        Order.Add(newOrder);
        CheckOrderLinesVisibility(newOrder);
    });
    var mainItem = Main.FirstOrDefault(m => m.Id == mainChart.Id);
    if (mainItem != null)
    {
        await Dispatcher.UIThread.InvokeAsync(() =>
        {
            mainItem.Quantity = mainChart.Quantity;
        });
    }
    UpdateOrderCollection();
}
private void CheckOrderLinesVisibility(OrderChart order)
{
```

```
    if (order.MaterialType == "Plate" && order.Status == "Complete")
    {
        FirstRackLinesVisibleForOrderComplete = true;
    }
    if (order.MaterialType == "Wire" && order.Status == "Complete")
    {
        SecondRackLinesVisibleForOrderComplete = true;
    }
    if (order.MaterialType == "Pipe" && order.Status == "Complete")
    {
        ThirdRackLinesVisibleForOrderComplete = true;
    }
    if (order.MaterialType == "Bag" && order.Status == "Complete")
    {
        FourthRackLinesVisibleForOrderComplete = true;
    }
}
private async void UpdateOrderCollection()
{
    using var context = new AppDBContext();
    var orders = context.OrderChart.ToList();
    await Dispatcher.UIThread.InvokeAsync(() =>
    {
        Order.Clear();
        foreach (var order in orders)
        {
            Order.Add(order);
        }
    });
}
private void UpdateOrdersOnQuantityChange(string productName, string materialType)
{
    using var context = new AppDBContext();
    var item = context.MainChart.FirstOrDefault(x => x.Name == productName && x.MaterialType
== materialType);
    if (item != null)
    {
        var inProcessOrders = context.OrderChart
            .Where(o => o.Name == productName && o.Status == "In process")
            .OrderBy(o => o.Data)
            .ToList();
        foreach (var order in inProcessOrders)
        {
            if (item.Quantity >= order.Quantity)
            {
                item.Quantity -= order.Quantity;
                order.Status = "Complete";
                Dispatcher.UIThread.InvokeAsync(() =>
                {
                    StatusBar = $"Order {order.Id} completed for item: {productName}";
                    CheckOrderLinesVisibility(order);
                }).Wait();
            }
        }
        context.SaveChanges();
        UpdateMainChart();
        UpdateOrderCollection();
    }
}
```

ДОДАТОК Г
Код замовлень потреб виробництва

```
// ProductionChart
private async Task AddProductionAsync()
{
    using var context = new AppDBContext();
    FirstRackLinesVisibleForProductionComplete = false;
    SecondRackLinesVisibleForProductionComplete = false;
    ThirdRackLinesVisibleForProductionComplete = false;
    FourthRackLinesVisibleForProductionComplete = false;
    var mainChart = context.MainChart.FirstOrDefault(m => m.Name == PName);
    if (mainChart == null)
    {
        await Dispatcher.UIThread.InvokeAsync(() =>
        {
            StatusBar = "Name not found in main chart.";
        });
        return;
    }
    string productionStatus;
    if (mainChart.Quantity < PQuantity)
    {
        productionStatus = "In process";
        await Dispatcher.UIThread.InvokeAsync(() =>
        {
            StatusBar = "Insufficient products in stock.";
        });
    }
    else
    {
        mainChart.Quantity -= PQuantity;
        productionStatus = "Complete";
        await Dispatcher.UIThread.InvokeAsync(() =>
        {
            StatusBar = "Production successfully added.";
        });
    }
    var newProduction = new ProductionChart
    {
        Data = DateTime.Now,
        Name = PName,
        MaterialType = mainChart.MaterialType,
        Quantity = PQuantity,
        Status = productionStatus
    };

    context.ProductionChart.Add(newProduction);
    await context.SaveChangesAsync();
    await Dispatcher.UIThread.InvokeAsync(() =>
    {
        Production.Add(newProduction);
        CheckProductionLinesVisibility(newProduction);
    });
    var mainItem = Main.FirstOrDefault(m => m.Id == mainChart.Id);
    if (mainItem != null)
    {
        await Dispatcher.UIThread.InvokeAsync(() =>
        {
            mainItem.Quantity = mainChart.Quantity;
        });
    }
    UpdateProductionCollection();
}
private void CheckProductionLinesVisibility(ProductionChart production)
{
```

```
    if (production.MaterialType == "Plate" && production.Status == "Complete")
    {
        FirstRackLinesVisibleForProductionComplete = true;
    }
    if (production.MaterialType == "Wire" && production.Status == "Complete")
    {
        SecondRackLinesVisibleForProductionComplete = true;
    }
    if (production.MaterialType == "Pipe" && production.Status == "Complete")
    {
        ThirdRackLinesVisibleForProductionComplete = true;
    }
    if (production.MaterialType == "Bag" && production.Status == "Complete")
    {
        FourthRackLinesVisibleForProductionComplete = true;
    }
}
private async void UpdateProductionCollection()
{
    using var context = new AppDBContext();
    var productions = context.ProductionChart.ToList();
    await Dispatcher.UIThread.InvokeAsync(() =>
    {
        Production.Clear();
        foreach (var production in productions)
        {
            Production.Add(production);
        }
    });
}
private void UpdateProductionOnQuantityChange(string productName, string materialType)
{
    using var context = new AppDBContext();
    var item = context.MainChart.FirstOrDefault(x => x.Name == productName && x.MaterialType
== materialType);
    if (item != null)
    {
        var inProcessProductions = context.ProductionChart
            .Where(p => p.Name == productName && p.Status == "In process")
            .OrderBy(p => p.Data)
            .ToList();
        foreach (var production in inProcessProductions)
        {
            if (item.Quantity >= production.Quantity)
            {
                item.Quantity -= production.Quantity;
                production.Status = "Complete";
                Dispatcher.UIThread.InvokeAsync(() =>
                {
                    StatusBar = $"Production {production.Id} completed for item:
{productName}";
                    CheckProductionLinesVisibility(production);
                }).Wait();
            }
        }
        context.SaveChanges();
        UpdateMainChart();
        UpdateProductionCollection();
    }
}
```

ДОДАТОК Д
Код сканеру для інвентаризації товару

```
//Camera In
private void InitializeINCamera()
{
    if (_captureIN != null)
    {
        _captureIN.Release();
        _captureIN.Dispose();
        _captureIN = null;
    }
    _captureIN = new VideoCapture(0);
    if (!_captureIN.IsOpened())
    {
        Dispatcher.UIThread.InvokeAsync(() =>
        {
            StatusBar = "Failed to initialize 'IN' camera.";
        }).Wait();
    }
}
public async Task StartINCameraAsync()
{
    await Task.Run(() => InitializeINCamera());
    if (!_captureIN.IsOpened())
    {
        await Dispatcher.UIThread.InvokeAsync(() =>
        {
            StatusBar = "Failed to open 'IN' camera.";
        });
        return;
    }
    _isCameraINRunning = true;
    await Dispatcher.UIThread.InvokeAsync(() =>
    {
        StatusBar = "'IN' camera started.";
    });
    await Task.Run(() =>
    {
        using (var mat = new Mat())
        {
            while (_isCameraINRunning)
            {
                lock (_cameraLock)
                {
                    try
                    {
                        _captureIN.Read(mat);
                        if (!mat.Empty())
                        {
                            using (var memoryStream = mat.ToMemoryStream())
                            {
                                var bitmap = new Bitmap(memoryStream);
                                Dispatcher.UIThread.InvokeAsync(() =>
                                {
                                    CameraINImage = bitmap;
                                }).Wait();
                            }
                            Point2f[] points;
                            var result = _qrCodeINDetector.DetectAndDecode(mat, out points);

                            if (result != null)
                            {
                                string decodedText = result;
                                if (!string.IsNullOrEmpty(decodedText))
                                {
                                    var parts = decodedText.Split(':');
                                    if (parts.Length == 2)
                                    {
                                        var productName = parts[0];
                                        var materialType = parts[1];
                                        IncrementProductQuantity(productName, materialType);
                                        Dispatcher.UIThread.InvokeAsync(() =>
                                        {
                                            StatusBar = $"Detected QR code: {decodedText}";
                                        }).Wait();
                                    }
                                    else
                                    {
                                        Dispatcher.UIThread.InvokeAsync(() =>
                                        {
                                            StatusBar = $"Invalid QR code format: {decodedText}";
                                        }).Wait();
                                    }
                                    Thread.Sleep(2000);
```

```
                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    Dispatcher.UIThread.InvokeAsync(() =>
                    {
                        StatusBar = $"Waiting for a QR code.";
                    }).Wait();
                }
            }
        }
    }
});
}
private void IncrementProductQuantity(string productName, string materialType)
{
    using var context = new AppDBContext();
    var item = context.MainChart.FirstOrDefault(x => x.Name == productName && x.MaterialType == materialType);
    if (item != null)
    {
        item.Quantity += 1;
        context.SaveChanges();
        Dispatcher.UIThread.InvokeAsync(() =>
        {
            StatusBar = $"Updated quantity for item: {productName} with material type: {materialType}";
            FirstRackLinesVisible = materialType == "Plate";
            SecondRackLinesVisible = materialType == "Wire";
            ThirdRackLinesVisible = materialType == "Pipe";
            FourthRackLinesVisible = materialType == "Bag";
        }).Wait();
        UpdateOrdersOnQuantityChange(productName, materialType);
        UpdateProductionOnQuantityChange(productName, materialType);
        context.SaveChanges();
        UpdateMainChart();
        UpdateOrderCollection();
        UpdateProductionCollection();
    }
    else
    {
        var newItem = new MainChart
        {
            Name = productName,
            Data = DateTime.Now,
            MaterialType = materialType,
            Quantity = 1,
            Rack = 1
        };
        context.MainChart.Add(newItem);
        context.SaveChanges();
        Dispatcher.UIThread.InvokeAsync(() =>
        {
            StatusBar = $"Created new item: {productName} with material type: {materialType}";
            FirstRackLinesVisible = materialType == "Plate";
            SecondRackLinesVisible = materialType == "Wire";
            ThirdRackLinesVisible = materialType == "Pipe";
            FourthRackLinesVisible = materialType == "Bag";
        }).Wait();
        UpdateOrdersOnQuantityChange(productName, materialType);
        UpdateProductionOnQuantityChange(productName, materialType);
    }
}
public async void StopINCamera()
{
    _isCameraINRunning = false;
    _captureIN?.Release();
    _captureIN?.Dispose();
    _captureIN = null;
    await Dispatcher.UIThread.InvokeAsync(() =>
    {
        CameraINImage = null;
        StatusBar = "'IN' camera stopped.";
    });
    UpdateMainChart();
    UpdateOrderCollection();
    UpdateProductionCollection();
}
```

ДОДАТОК Е
Код для видалення товарів

```
//Camera Delete
private void InitializeDeleteCamera()
{
    if (_captureDelete != null)
    {
        _captureDelete.Release();
        _captureDelete.Dispose();
        _captureDelete = null;
    }
    _captureDelete = new VideoCapture(0);
    if (!_captureDelete.IsOpened())
    {
        Dispatcher.UIThread.InvokeAsync(() =>
        {
            StatusBar = "Failed to initialize 'DELETE' camera.";
        }).Wait();
    }
}
public async Task StartDeleteCameraAsync()
{
    await Task.Run(() => InitializeDeleteCamera());
    if (!_captureDelete.IsOpened())
    {
        await Dispatcher.UIThread.InvokeAsync(() =>
        {
            StatusBar = "Failed to open 'DELETE' camera.";
        });
        return;
    }
    _isDeleteCameraRunning = true;
    await Dispatcher.UIThread.InvokeAsync(() =>
    {
        StatusBar = "'DELETE' camera started.";
    });
    await Task.Run(() =>
    {
        using (var mat = new Mat())
        {
            while (_isDeleteCameraRunning)
            {
                lock (_cameraLock)
                {
                    try
                    {
                        _captureDelete.Read(mat);
                        if (!mat.Empty())
                        {
                            using (var memoryStream = mat.ToMemoryStream())
                            {
                                var bitmap = new Bitmap(memoryStream);
                                Dispatcher.UIThread.InvokeAsync(() =>
                                {
                                    CameraDeleteImage = bitmap;
                                }).Wait();
                            }
                            Point2f[] points;
                            var result = _qrCodeDeleteDetector.DetectAndDecode(mat, out points);
                            if (result != null)
                            {
                                string decodedText = result;
                                if (!string.IsNullOrEmpty(decodedText))
                                {
                                    var parts = decodedText.Split(':');
                                    if (parts.Length == 2)
                                    {
                                        var productName = parts[0];
                                        var materialType = parts[1];
                                        DeleteDataFromDatabase(productName, materialType);
                                        Dispatcher.UIThread.InvokeAsync(() =>
                                        {
                                            StatusBar = $"Detected QR code: {decodedText}";
```

```
                                    }).Wait();
                                }
                                else
                                {
                                    Dispatcher.UIThread.InvokeAsync(() =>
                                    {
                                        StatusBar = $"Invalid QR code format: {decodedText}";
                                    }).Wait();
                                }
                            }
                        }
                    }
                    catch (Exception ex)
                    {
                        Dispatcher.UIThread.InvokeAsync(() =>
                        {
                            StatusBar = $"Waiting for a QR code.";
                        }).Wait();
                    }
                }
            }
        }
    });
}
private void DeleteDataFromDatabase(string productName, string materialType)
{
    using var context = new AppDBContext();
    var item = context.MainChart.FirstOrDefault(x => x.Name == productName && x.MaterialType ==
materialType);
    if (item != null)
    {
        context.MainChart.Remove(item);
        context.SaveChanges();
        Dispatcher.UIThread.InvokeAsync(() =>
        {
            StatusBar = $"Deleted item: {productName} with material type: {materialType}";
        }).Wait();
    }
    else
    {
        Dispatcher.UIThread.InvokeAsync(() =>
        {
            StatusBar = $"Item not found: {productName} with material type: {materialType}";
        }).Wait();
    }
    UpdateMainChart();
}
public async void StopDeleteCamera()
{
    _isDeleteCameraRunning = false;
    _captureDelete?.Release();
    _captureDelete?.Dispose();
    _captureDelete = null;
    await Dispatcher.UIThread.InvokeAsync(() =>
    {
        CameraDeleteImage = null;
        StatusBar = "'DELETE' camera stopped.";
    });
    UpdateMainChart();
}
```

ДОДАТОК Ж
Код генерації QR-коду

```
//QR-code
 public void GenerateQRCode()
 {
     if (string.IsNullOrEmpty(Text1) || string.IsNullOrEmpty(Text2))
     {
         StatusBar = "One or both text fields are null or empty.";
         return;
     }
     var textToEncode = $"{Text1}:{Text2}";
     var writer = new BarcodeWriterPixelData
     {
         Format = BarcodeFormat.QR_CODE,
         Options = new EncodingOptions
         {
             Height = 200,
             Width = 200
         }
     };
     try
     {
         var pixelData = writer.Write(textToEncode);
         var bitmap = CreateBitmapFromPixelData(pixelData);
         if (bitmap == null)
         {
             StatusBar = "Failed to create bitmap from pixel data.";
             return;
         }
         QRCode = bitmap;
         SaveQRCodeToFile(pixelData, textToEncode);
     }
     catch (Exception ex)
     {
         StatusBar = $"Error generating QR code: {ex.Message}";
     }
 }
 public Bitmap CreateBitmapFromPixelData(PixelData pixelData)
 {
     try
     {
         using (var writeableBitmap = new WriteableBitmap(
             new PixelSize(pixelData.Width, pixelData.Height),
             new Vector(96, 96), // DPI
             PixelFormat.Bgra8888,
             AlphaFormat.Premul))
         {
             using (var lockedBuffer = writeableBitmap.Lock())
             {
                 System.Runtime.InteropServices.Marshal.Copy(pixelData.Pixels, 0,
lockedBuffer.Address, pixelData.Pixels.Length);
             }
             using (var stream = new MemoryStream())
             {
                 writeableBitmap.Save(stream);
                 stream.Seek(0, SeekOrigin.Begin);
                 return new Bitmap(stream);
             }
         }
```

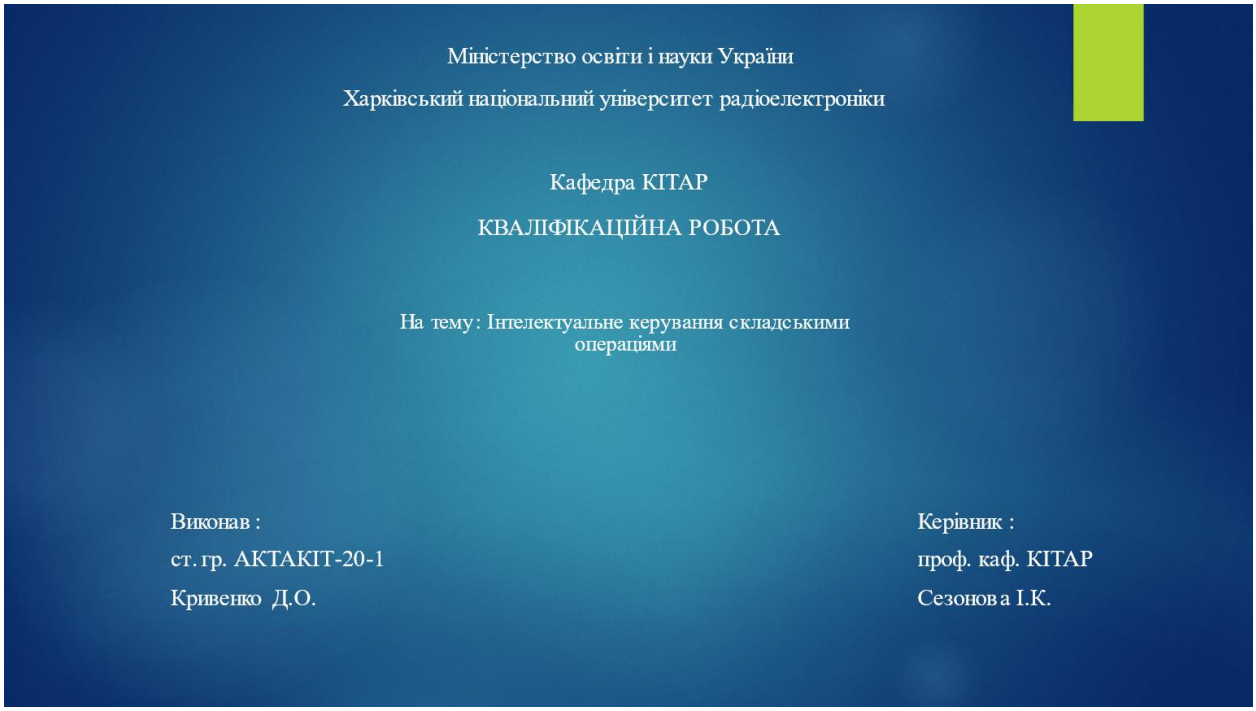
```
        }
        catch (Exception ex)
        {
            StatusBar = $"Error creating bitmap: {ex.Message}";
            return null;
        }
    }
    public void SaveQRCodeToFile(PixelData pixelData, string textToEncode)
    {
        string folderPath = "D:\\all\\study\\VS\\Diplom directorys\\QRcode";
        if (!Directory.Exists(folderPath))
        {
            Directory.CreateDirectory(folderPath);
        }
        string sanitizedText =
string.Concat(textToEncode.Split(Path.GetInvalidFileNameChars()));
        string fileName = $"{sanitizedText}.jpg";
        string filePath = Path.Combine(folderPath, fileName);
        try
        {
            using (var writeableBitmap = new WriteableBitmap(
                new PixelSize(pixelData.Width, pixelData.Height),
                new Vector(96, 96), // DPI
                PixelFormat.Bgra8888,
                AlphaFormat.Premul))
            {
                using (var lockedBuffer = writeableBitmap.Lock())
                {
                    System.Runtime.InteropServices.Marshal.Copy(pixelData.Pixels, 0,
lockedBuffer.Address, pixelData.Pixels.Length);
                }
                using (var fileStream = new FileStream(filePath, FileMode.Create))
                {
                    writeableBitmap.Save(fileStream);
                }
            }
            StatusBar = $"QR-code saved: {filePath}";
        }
        catch (Exception ex)
        {
            StatusBar = $"Error saving QR code to file: {ex.Message}";
        }
    }
```

ДОДАТОК И
Код для генерації звітів та аналізу даних

```
//Analysis
public async void GenerateReport()
{
    var reportDirectory = "D:\\all\\study\\VS\\Diplom directorys\\Warehouse report";
    if (!Directory.Exists(reportDirectory))
    {
        Directory.CreateDirectory(reportDirectory);
    }
    var reportPath = Path.Combine(reportDirectory, "WarehouseReport.txt");

    using var context = new AppDBContext();
    var reportData = await context.MainChart.ToListAsync();

    var report = new StringBuilder();
    report.AppendLine("Warehouse report:");
    foreach (var item in reportData)
    {
        report.AppendLine($"Product: {item.Name}, Quantity: {item.Quantity}, Data:
{item.Data}");
    }

    await File.WriteAllTextAsync(reportPath, report.ToString());
    StatusBar = "The report is generated and saved to the WarehouseReport.txt file.";
}

public async void ShowAnalytics()
{
    using var context = new AppDBContext();
    var analyticsData = await context.MainChart.ToListAsync();

    var totalProducts = analyticsData.Sum(x => x.Quantity);
    var distinctProducts = analyticsData.Select(x => x.Name).Distinct().Count();

    StatusBar = $"Total products in warehouse: {totalProducts}, Various names:
{distinctProducts}";
}
```

ДОДАТОК К

Демонстраційний матеріал у вигляді презинтації



ДОДАТОК Л

Відомість кваліфікаційної роботи

	Позначення	Найменування	Дод. Відомості
		Текстові документи	
1	ГЮИК.505900.021ПЗ	Пояснювальна записка	А4, 93 с.
		Додаткові матеріали	
2		Демонстраційний матеріал	А4, 13 с.
		у вигляді презентації	

					ГЮИК. 505900.021ВД			
Змін.	Арк.	Номер докум.	Підп.	Дата				
Розроб.		Кривенко Д.О.		15.06.2024	Інтелектуальне керування складськими операціями	Літера	Аркуш	Аркушів
Перевір.		Сезонова І.К.				Н	1	1
Н.контр.		Стародубцев М.Г.				ХНУРЕ Кафедра КІТАМ		
Затв.		Невлюдов І.Ш.						