COIL YARD MANAGEMENT SYSTEM

NAME OF THE TRAINEE : ANANT SHRIVASTAVA

NAME OF THE COMPANY : TATA BLUESCOPE STEEL

NAME OF THE SUPERVISOR : Mr. MANISH UPADHAYAY

FIELD OF TRAINING : AUTOMATION/IT

CERTIFICATE

This is to certify that the project work entitled **YARD MANAGEMENT SYSTEM** is a bonafide work done and submitted by **ANANT SHRIVASTAVA**, **in** partial fulfilment of the requirement for the award of the degree of Computer Science Engineering by **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, CHENNAI**.

This project was successfully created & submitted in **TATA BLUESCOPE STEEL.**

ACKNOWLEDGEMENT

I wish to record our heartfelt gratitude and sincere thanks to Mr**.** **Manish Upadhayay**, Dy. Head of Automation Department, TATA BLUESCOPE STEEL for his kind support and inspiration given to me till the end of my project.

Thanks to all senior members Abhishek Sir (Trainer), Barun sir, Chandan sir, Krishna sir for their constant support and encouragement given throughout the development of the project. We are very thankful sincerely they taught all about new software asp.net & MySQL & help me to complete us this project. Our heartly thanks for their patient & kind behaviour that help me to learn a very new thing in this organization.

Last but not the least our sincere thanks to my parents, family members and friends for their continuous support, inspiration, and encouragement without which this project would not have been success.

TABLE OF CONTENTS

Contents

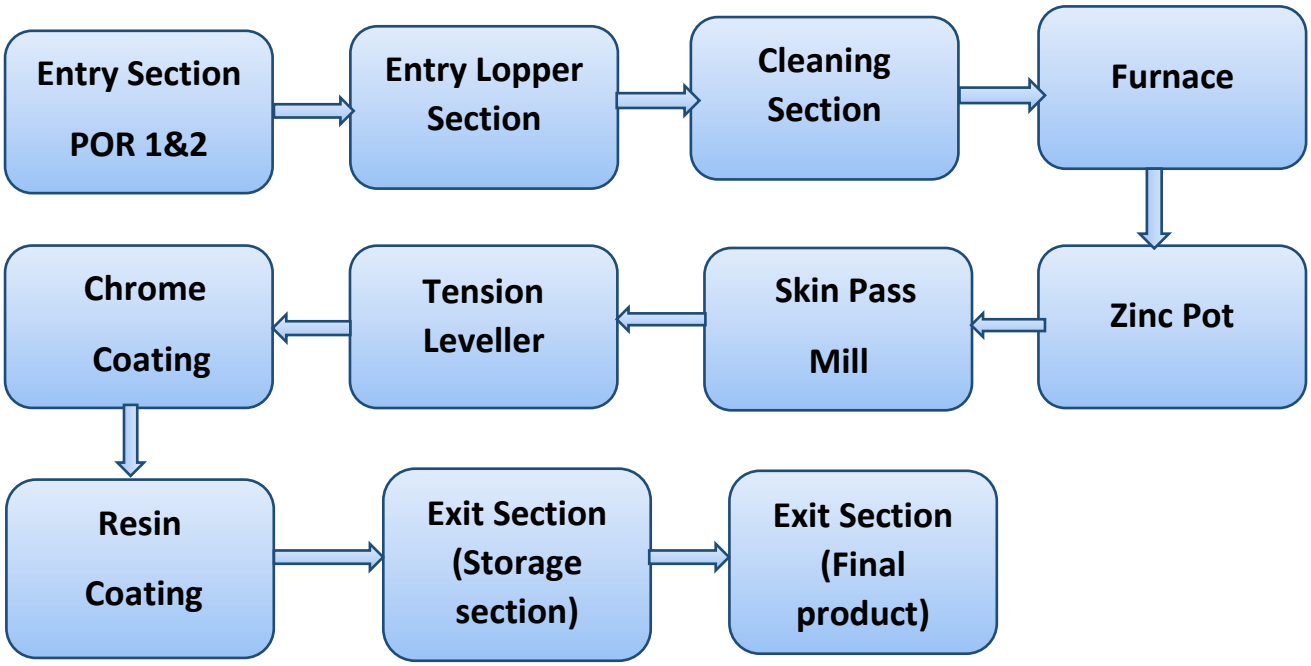
INTRODUCTION ABOUT TBSPL

TATA BLUESCOPE STEEL Pvt. Ltd., a joint venture between **TATA STEEL** Ltd. INDIA, and **BLUESCOPE STEEL** Ltd. AUSTRALIA established in 2005, proposes to set up a major state-of-the-art metallic coating and painting facility with three major businesses: Coated Steel, Roof & Wall Cladding Products and Pre-engineered Building Solutions. The company aims to actively pursue market development, of coated steel and its application by developing, customer oriented, reliable, and efficient solutions. Tata BlueScope Steel is headquartered in Pune and has its manufacturing units in Jamshedpur, Sriperumbudur, Bhiwandi and Pune. Tata BlueScope Steel has an annual metallic coating capacity of 250,000 tons and colour coating capacity of 150,000 tons. The proposed coated steel complex will be situated adjacent to TATA STEEL's existing steel work.

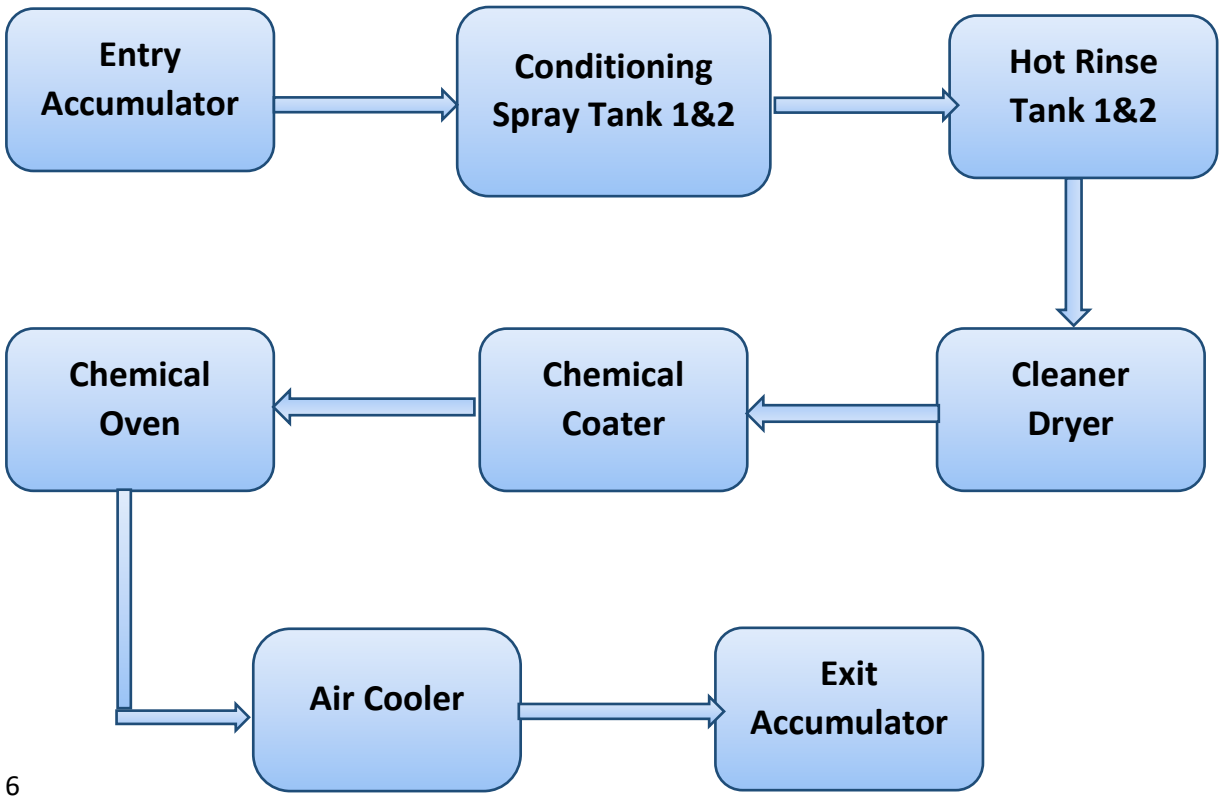
There are four main levels in automation part:

- MCL (Metal Coating Line)
- CCL (Colour Coating Line)
- SRL (Slitting & Re-Coiling Line)
- PL (Pack Line)

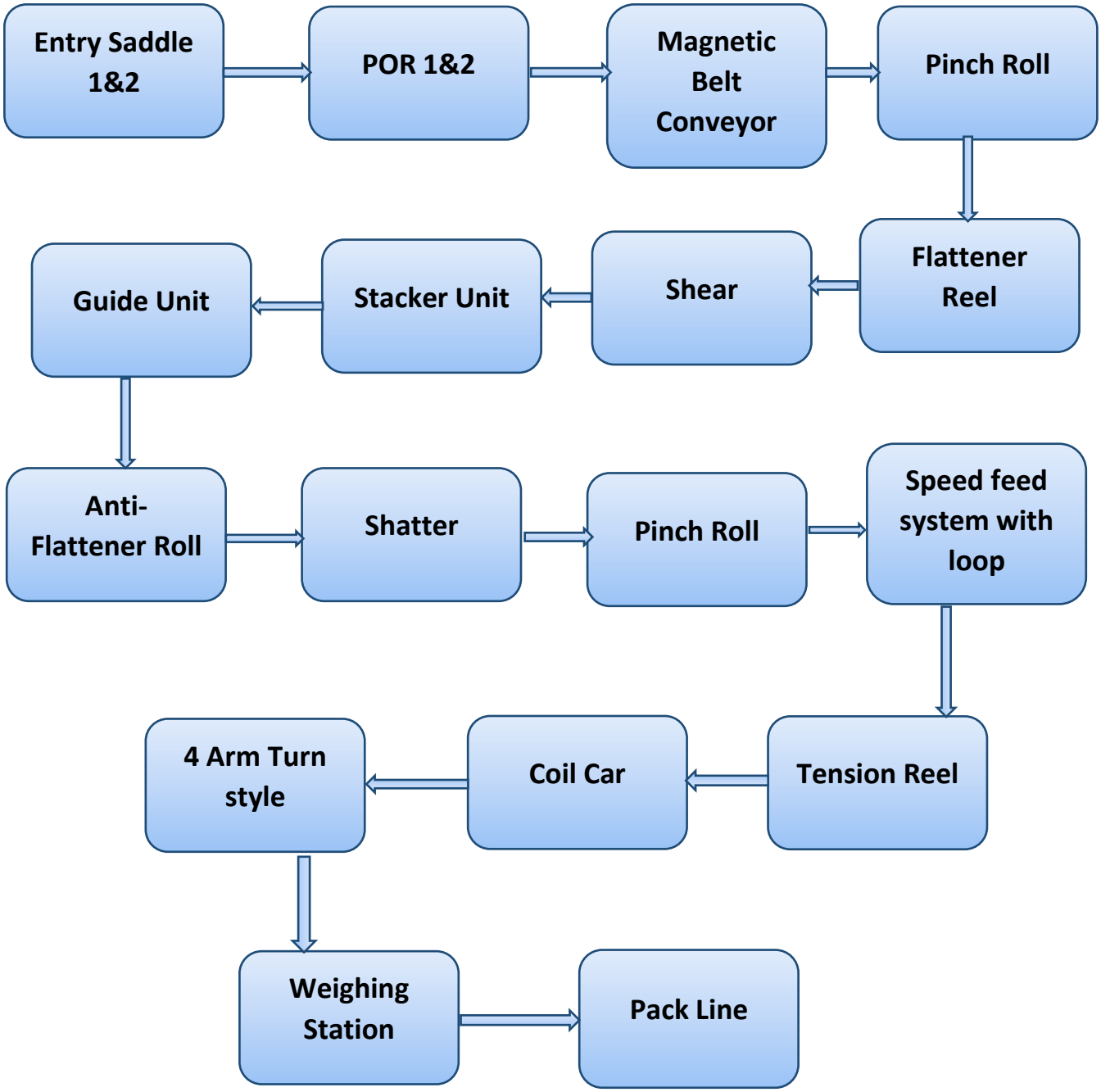
MCL


Entry Section POR 1&2
Entry Lopper Section
Cleaning Section
Furnace
Zinc Pot
Skin Pass Mill
Tension Leveller
Chrome Coating
Resin Coating
Exit Section (Storage section)
Exit Section (Final product)

CCL


Entry Accumulator
Conditioning Spray Tank 1&2
Hot Rinse Tank 1&2
Cleaner Dryer
Chemical Coater
Chemical Oven
Air Cooler
Exit Accumulator

SRL


Entry Saddle 1&2
POR 1&2
Magnetic Belt Conveyor
Pinch Roll
Flattener Reel
Shear
Stacker Unit
Guide Unit
Anti-Flattener Roll
Shatter
Pinch Roll
Speed feed system with loop
Tension Reel
Coil Car
4 Arm Turn style
Weighing Station
Pack Line

CURRENT PROCESS USED

TATA BLUESCOPE STEEL is a major steel producer that operates steel mills and manufacturing facilities around the world. They produce steel coils and other steel products that are stored and transported through their yards. In their **Yard Management System**, coils are tracked and managed using a combination of barcode labels, RFID tags, and other identification methods. Each coil is assigned a unique ID number that is printed on a label and/or encoded in an RFID tag attached to the coil. When a coil arrives at the yard, its ID, dimensions, weight, grade, and other key data are scanned into the yard management system. The system tracks the coil's location in the yard storage area based on its ID. Cranes and other yard equipment are used to move the coils around the yard. The crane operators use the coil IDs to identify which coils to pick up and where to place them. The Yard management software integrates with **TATA BLUESCOPE's** other enterprise systems to share data on coil inventory, production schedules, and customer orders. This allows them to efficiently manage the flow of coils from the mills to the customers. When a coil is loaded onto a truck for delivery, its ID is scanned again and the system records the departure time and destination. This provides full traceability of the coil's journey through the yard. The use of automated data capture and yard management software helps BlueScope optimize their coil storage, reduce handling time and costs, and ensure accurate delivery of the right coils to customers.

The problems with the current system:

Lack of Control: There is no control over material and vehicle movements. Unauthorized access can lead to security risks.

No Authentication Process: Manual entries in registers are not authenticated, leading to unreliable data.

Inefficient Process: Noting data in registers is laborious and time-consuming.

The Yard Management System will replace the manual entry process with a sophisticated internet-based system to overcome the above problems. It makes the process faster and more secure.

MOTO TO BUILD THIS PROJECT

The motto or guiding principle for a Yard Management System (YMS) could be something along these lines:

"Optimizing Yard Efficiency, Enhancing Supply Chain Visibility"

The key aspects that a YMS motto could emphasize are:

1. **Optimize Yard Operations**: The primary goal of the Yard Management System (YMS) is to optimize the operations and utilization of the storage yard. This includes efficient use of space, streamlining material handling, and minimizing delays to enhance overall productivity and efficiency.

2. **Real-Time Visibility and Control**: The YMS provides real-time visibility into yard activities and inventory, improving decision-making, planning, and coordination across the supply chain. The ability to control and monitor both planned and unplanned activities reduces risks and enhances security.

3. **Integrated Supply Chain Management**: The YMS seamlessly integrates with other supply chain systems, such as transportation management, warehouse management, and ERP. This integration enables a holistic view of end-to-end supply chain operations, improving overall efficiency.

4. **Enhanced Efficiency and Productivity**: The YMS drives increased efficiency and productivity in the yard by reducing manual effort and errors. Automation, data-driven decision-making, and process optimization are key components to achieving this goal, ensuring smoother yard operations.

5. **Compliance and Safety**: The YMS ensures compliance with industry regulations and maintains high safety standards in the yard. Proper tracking, monitoring, and reporting of yard activities contribute to maintaining a safe and compliant operational environment. Notifications for scheduled material movements and live status updates further enhance yard management security.

By encapsulating these key aspects, a YMS motto could inspire the development and implementation of a system that truly optimizes yard operations and enhances the overall supply chain performance.

DETAILS OF PROJECT UNDERTAKEN

Coil Movement App

The Coil Movement App is a Windows Forms application that simulates the movement of coils from a saddle to a truck using a crane. The Yard Management System helps the organization manage yard operations effectively. It tracks material movements, vehicle entries, and exits, ensuring the safety and efficiency of yard operations.

Features

1. Coil Saddle: The application displays the coils currently on the saddle.

2. Crane Movement: The user can click the "GO" button to simulate the crane picking up a coil from the saddle and placing it on the yard.

4. Coil Information: The application displays the details of the coil that is currently being moved, including the coil ID, name, arrival time, and departure time.

5. Yard Display: Once all the coils have been moved to the truck, the application displays a message indicating that the coils have been moved to the yard.

Code Structure

The Coil Movement App is composed of two main files:

1. Program.cs: This file contains the entry point of the application and creates an instance of the `Form1` class.

2. Form1.cs: This file contains the implementation of the `Form1` class, which is the main form of the application. It includes the following:

- `saddleCoils` and `truckCoils` lists to store the coils on the saddle and the truck, respectively.

- The `buttonMoveCrane_Click` event handler, which is called when the "Go" button is clicked. This method simulates the movement of a coil from the saddle to the truck.

- The `Coil` class, which represents a single coil with properties for the coil ID, name, arrival time, and departure time.

3. Form1.Designer.cs: This file includes the code for the design of the front-end of the system. It includes the properties of the resources used in the making of this system.

Uses

Yard Management System

A Yard Management System (YMS) is a software application used in logistics and supply chain management to optimize the operations of a storage yard or terminal. It is typically used in industries such as manufacturing, warehousing, and transportation to manage the movement and storage of goods, materials, and equipment.

Uses of a Yard Management System

1. **Inventory Management:** A YMS helps companies track and manage the inventory of goods, materials, and equipment stored in the yard. This includes monitoring the location, quantity, and status of each item.

2. **Yard Optimization:** The system helps optimize the layout and utilization of the yard, ensuring efficient use of available space and resources. This includes managing the placement and movement of vehicles, containers, and other assets.

3. **Scheduling and Coordination:** A YMS helps coordinate the arrival, departure, and movement of vehicles, containers, and other assets within the yard. This includes scheduling and managing loading, unloading, and storage activities.

4. **Visibility and Reporting:** The system provides real-time visibility into the yard's operations, allowing managers to monitor and analyse performance metrics, generate reports, and make informed decisions.

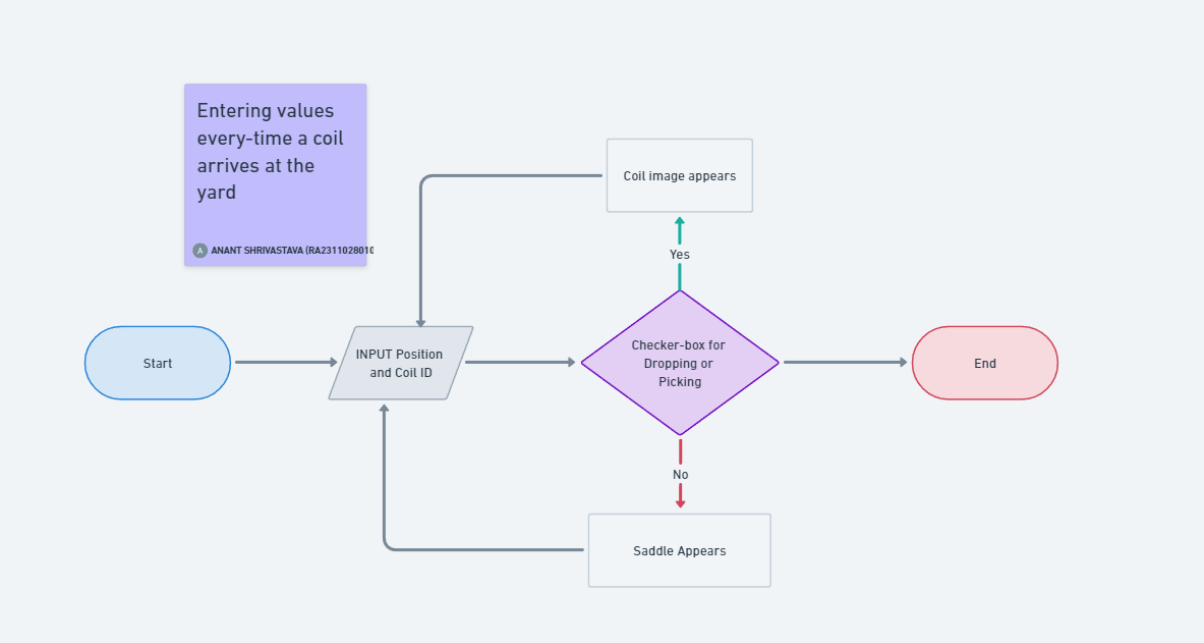
5. **Automation and Efficiency:** A YMS can automate various yard operations, such as gate entry and exit, container tracking, and yard equipment management, improving overall efficiency and reducing manual labour.

6. **Compliance and Safety:** The system helps ensure compliance with industry regulations and safety standards, such as managing hazardous materials, tracking equipment maintenance, and monitoring worker safety.

7. **Integration with Other Systems**: A YMS can integrate with other enterprise systems, such as transportation management systems, warehouse management systems, and enterprise resource planning (ERP) systems, to provide a comprehensive view of the supply chain.

By implementing a Yard Management System, companies can improve the efficiency of their yard operations, reduce costs, and enhance customer service by ensuring the timely and accurate movement of goods, materials, and equipment.

DATA FLOW DIAGRAM OF YMS



TECHNOLOGIES USED

- VISUAL STUDIO 2010
- C#
- .NET
- MySQL

- **VISUAL STUDIO 2010**

Visual Basic 2010 was launched by Microsoft in 2010. Microsoft Visual Studio is an integrated development environment (IDE) which is used to develop computer programs for Microsoft Windows, as well as web sites, web applications and web services. It supports different programming languages like C, C++ and C++/CLI, Visual Basic .NET, C#, and F#.

- **C#:**

C# is pronounced "C sharp." C# is an object-oriented programming language and part of the .NET family from Microsoft. C# is very similar to c ++ and java. C# is developed by Microsoft and works on the windows and web platforms.

- **.NET FRAME WORK:**

The .NET frame work (pronounced "dot net") is a software frame work that runs primarily on Microsoft windows. It includes a large library and support several programming languages which allow language interoperability (each language can use code written in other languages). The .NET library is available to all the programming languages that .NET supports. Programs wri9tten for the .NET frame

work execute in a software environment, known as the common language runtime (CLR), an application virtual machine that provides important services such as security, memory management, and exception handling. The class library and the CLR together constitute the .NET framework.

- **MySQL:**

A MySQL **database** is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management. In general, a **server** reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

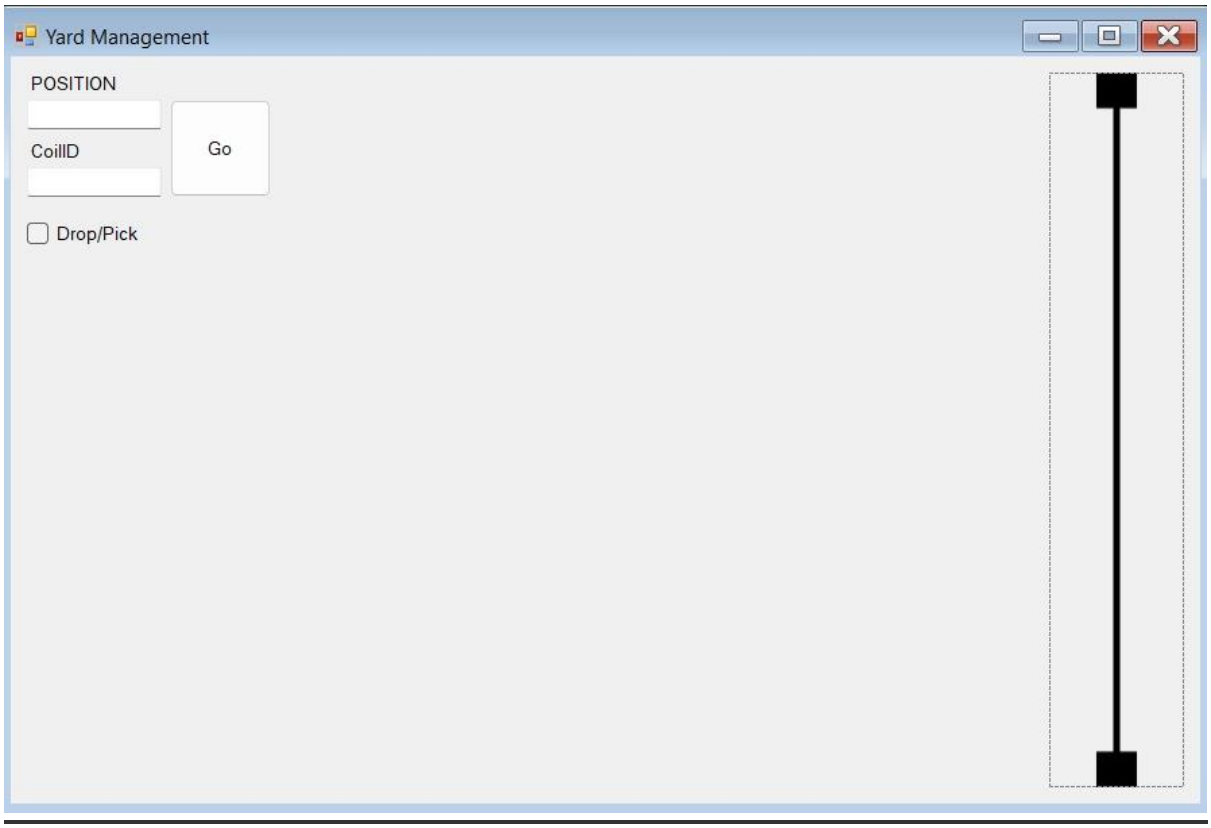
MySQL Database is the first database designed for enterprise grid computing, the most flexible and cost-effective way to manage information and applications. Enterprise grid computing creates large pools of industry-standard, modular storage, and servers. With this architecture, each new system can be rapidly provisioned from the pool of components.

The database has **logical structures** and **physical structures**. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

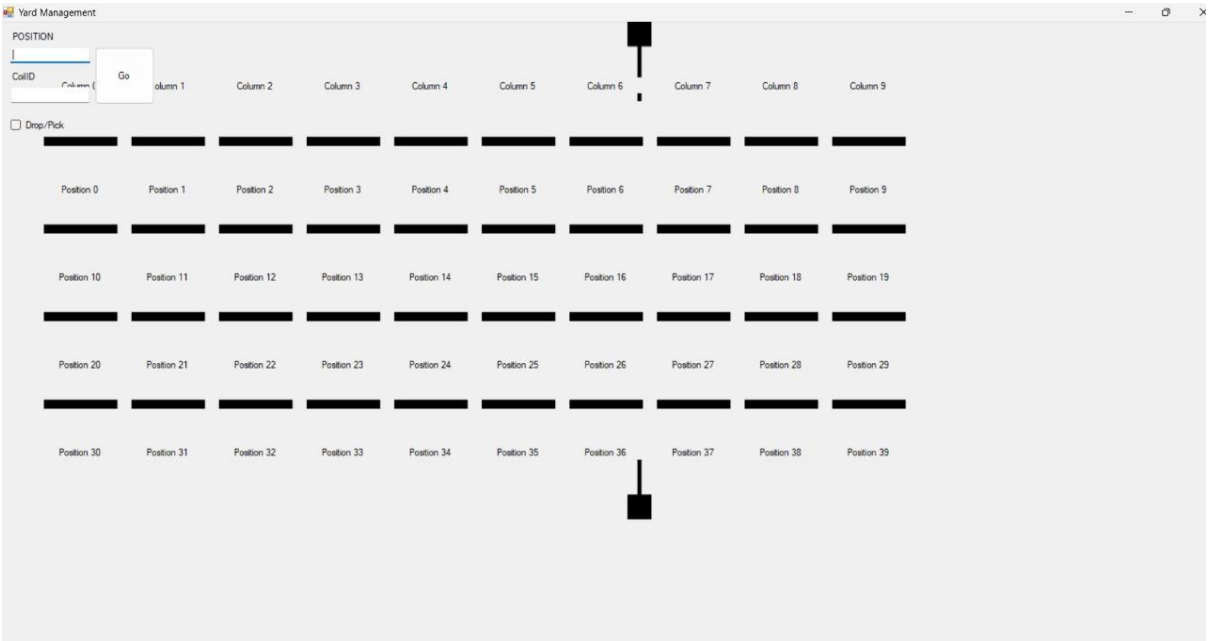
DOTNET

Along with DOTNET, visual studio was used as front end for MySQL

INITIAL DESIGN



OPENING SCREEN



Coding for Design Page:

Form.cs :

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
namespace YardManagement
{
 public partial class Form1 : Form
{
    private Dictionary<int, PictureBox> pictureBoxDictionary = new Dictionary<int, PictureBox>();
      private Dictionary<int, Label> labelDictionary = new Dictionary<int, Label>();
      private Point targetPosition;
      private System.Windows.Forms.TextBox textBoxCoilId;
      private System.Windows.Forms.TextBox textBoxLabel;
      private System.Windows.Forms.Button buttonGo;
      private System.Windows.Forms.PictureBox crane;
      private System.Windows.Forms.CheckBox checkBoxChangeImage;
      private System.Windows.Forms.Label labelCoilId;
      private System.Windows.Forms.Label labelLabel;

      public Form1()
      {
          InitializeComponent();
          InitializeGrid();
          crane.SendToBack(); // Send crane to back after initializing the grid
      }

      private void InitializeGrid()
      {
          int pictureBoxWidth = 100;
          int pictureBoxHeight = 100;
          int spacing = 10;

          // Create column number labels
          for (int col = 0; col < 10; col++)
          {
              Label columnLabel = new Label
```

```
        {
            Name = $"columnLabel{col}",
            Size = new Size(pictureBoxWidth, 20),
            Location = new Point(50 + (col * (pictureBoxWidth + spacing)), 70), // Adjust the y position as needed
            Text = $"Column {col}",
            TextAlign = ContentAlignment.MiddleCenter
        };
        this.Controls.Add(columnLabel);
    }

    for (int row = 0; row < 4; row++)
    {
        for (int col = 0; col < 10; col++)
        {
            int index = row * 10 + col;

            PictureBox pictureBox = new PictureBox
            {
                Name = $"pictureBox{index}",
                Size = new Size(pictureBoxWidth, pictureBoxHeight),
                Location = new Point(50 + (col * (pictureBoxWidth + spacing)), 100 + (row * (pictureBoxHeight + spacing))),
                Image = Properties.Resources.Cut_back,
                SizeMode = PictureBoxSizeMode.StretchImage
            };

            Label label = new Label
            {
                Name = $"label{index}",
                Size = new Size(pictureBoxWidth, 20),
                Location = new Point(50 + (col * (pictureBoxWidth + spacing)), 100 + pictureBoxHeight + (row *
(pictureBoxHeight + spacing))),
                Text = $"Position {index}",
                TextAlign = ContentAlignment.MiddleCenter
            };

            pictureBoxDictionary.Add(index, pictureBox);
            labelDictionary.Add(index, label);
            this.Controls.Add(pictureBox);
            this.Controls.Add(label);
        }
    }
```

```
}

private void ButtonGo_Click(object sender, EventArgs e)
{
    if (int.TryParse(textBoxCoilId.Text, out int id) && pictureBoxDictionary.ContainsKey(id))
    {
        targetPosition = pictureBoxDictionary[id].Location;
        MoveCrane(id);
    }
    else
    {
        MessageBox.Show("Please enter a valid coil ID.");
    }
}

private void MoveCrane(int id)
{
    Timer timer = new Timer { Interval = 10 };
    timer.Tick += (sender, e) =>
    {
        int step = 5; // step size

        // Move crane horizontally
        if (crane.Location.X != targetPosition.X)
        {
            int direction = targetPosition.X > crane.Location.X ? 1 : -1;
            int remainingDistance = Math.Abs(targetPosition.X - crane.Location.X);
            crane.Left += direction * Math.Min(step, remainingDistance);
        }

        else
        {
            timer.Stop();
            crane.Location = new Point(targetPosition.X, crane.Location.Y);

            if (checkBoxChangeImage.Checked)
            {
                pictureBoxDictionary[id].Image = Properties.Resources.Coil;
            }
            else
            {
```

```
                    pictureBoxDictionary[id].Image = Properties.Resources.Cut_back;
                }

                if (!string.IsNullOrWhiteSpace(textBoxLabel.Text))
                {
                    labelDictionary[id].Text = textBoxLabel.Text;
                }
            }
        };
        timer.Start();
    }
  }
}
```

Form.Designer.cs:

```
namespace YardManagement
{
    partial class Form1
    {
        private System.ComponentModel.IContainer components = null;

        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        private void InitializeComponent()
        {
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(Form1));
            this.textBoxCoilId = new System.Windows.Forms.TextBox();
            this.textBoxLabel = new System.Windows.Forms.TextBox();
            this.buttonGo = new System.Windows.Forms.Button();
            this.crane = new System.Windows.Forms.PictureBox();
            this.checkBoxChangeImage = new System.Windows.Forms.CheckBox();
```

```
this.labelCoilId = new System.Windows.Forms.Label();
this.labelLabel = new System.Windows.Forms.Label();
((System.ComponentModel.ISupportInitialize)(this.crane)).BeginInit();
this.SuspendLayout();
//
// textBoxCoilId
//
resources.ApplyResources(this.textBoxCoilId, "textBoxCoilId");
this.textBoxCoilId.Name = "textBoxCoilId";
//
// textBoxLabel
//
resources.ApplyResources(this.textBoxLabel, "textBoxLabel");
this.textBoxLabel.Name = "textBoxLabel";
//
// buttonGo
//
resources.ApplyResources(this.buttonGo, "buttonGo");
this.buttonGo.Name = "buttonGo";
this.buttonGo.UseVisualStyleBackColor = true;
this.buttonGo.Click += new System.EventHandler(this.ButtonGo_Click);
//
// crane
//
this.crane.Image = global::YardManagement.Properties.Resources.pngegg;
resources.ApplyResources(this.crane, "crane");
this.crane.Name = "crane";
this.crane.TabStop = false;
//
// checkBoxChangeImage
//
resources.ApplyResources(this.checkBoxChangeImage, "checkBoxChangeImage");
this.checkBoxChangeImage.Name = "checkBoxChangeImage";
this.checkBoxChangeImage.UseVisualStyleBackColor = true;
//
// labelCoilId
//
resources.ApplyResources(this.labelCoilId, "labelCoilId");
this.labelCoilId.Name = "labelCoilId";
//
// labelLabel
```

```
            //
            resources.ApplyResources(this.labelLabel, "labelLabel");
            this.labelLabel.Name = "labelLabel";
            //
            // Form1
            //
            resources.ApplyResources(this, "$this");
            this.Controls.Add(this.labelLabel);
            this.Controls.Add(this.labelCoilId);
            this.Controls.Add(this.checkBoxChangeImage);
            this.Controls.Add(this.buttonGo);
            this.Controls.Add(this.textBoxLabel);
            this.Controls.Add(this.textBoxCoilId);
            this.Controls.Add(this.crane);
            this.Name = "Form1";
            ((System.ComponentModel.ISupportInitialize)(this.crane)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

    }
}
```

Resource.resx:

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <!--
    Microsoft ResX Schema

    Version 2.0

    The primary goals of this format is to allow a simple XML format
    that is mostly human readable. The generation and parsing of the
    various data types are done through the TypeConverter classes
    associated with the data types.

    Example:
```

... ado.net/XML headers & schema ...

<resheader name="resmimetype">text/microsoft-resx</resheader>

<resheader name="version">2.0</resheader>

<resheader name="reader">System.Resources.ResXResourceReader, System.Windows.Forms, ...</resheader>

<resheader name="writer">System.Resources.ResXResourceWriter, System.Windows.Forms, ...</resheader>

<data name="Name1"><value>this is my long string</value><comment>this is a comment</comment></data>

<data name="Color1" type="System.Drawing.Color, System.Drawing">Blue</data>

<data name="Bitmap1" mimetype="application/x-microsoft.net.object.binary.base64">

<value>[base64 mime encoded serialized .NET Framework object]</value>

</data>

<data name="Icon1" type="System.Drawing.Icon, System.Drawing" mimetype="application/x-microsoft.net.object.bytearray.base64">

<value>[base64 mime encoded string representing a byte array form of the .NET Framework object]</value>

<comment>This is a comment</comment>

</data>

There are any number of "resheader" rows that contain simple
name/value pairs.

Each data row contains a name, and value. The row also contains a
type or mimetype. Type corresponds to a .NET class that support
text/value conversion through the TypeConverter architecture.
Classes that don't support this are serialized and stored with the
mimetype set.

The mimetype is used for serialized objects, and tells the
ResXResourceReader how to depersist the object. This is currently not
extensible. For a given mimetype the value must be set accordingly:

Note - application/x-microsoft.net.object.binary.base64 is the format
that the ResXResourceWriter will generate, however the reader can
read any of the formats listed below.

mimetype: application/x-microsoft.net.object.binary.base64
value : The object must be serialized with
: System.Runtime.Serialization.Formatters.Binary.BinaryFormatter
: and then encoded with base64 encoding.

mimetype: application/x-microsoft.net.object.soap.base64
value : The object must be serialized with
: System.Runtime.Serialization.Formatters.Soap.SoapFormatter
: and then encoded with base64 encoding.

```
    mimetype: application/x-microsoft.net.object.bytearray.base64
    value   : The object must be serialized into a byte array
            : using a System.ComponentModel.TypeConverter
            : and then encoded with base64 encoding.
    -->
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
              </xsd:sequence>
              <xsd:attribute name="name" use="required" type="xsd:string" />
              <xsd:attribute name="type" type="xsd:string" />
              <xsd:attribute name="mimetype" type="xsd:string" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="assembly">
            <xsd:complexType>
              <xsd:attribute name="alias" type="xsd:string" />
              <xsd:attribute name="name" type="xsd:string" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="data">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" msdata:Ordinal="1" />
                <xsd:element name="comment" type="xsd:string" minOccurs="0" msdata:Ordinal="2" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required" msdata:Ordinal="1" />
              <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
              <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
```

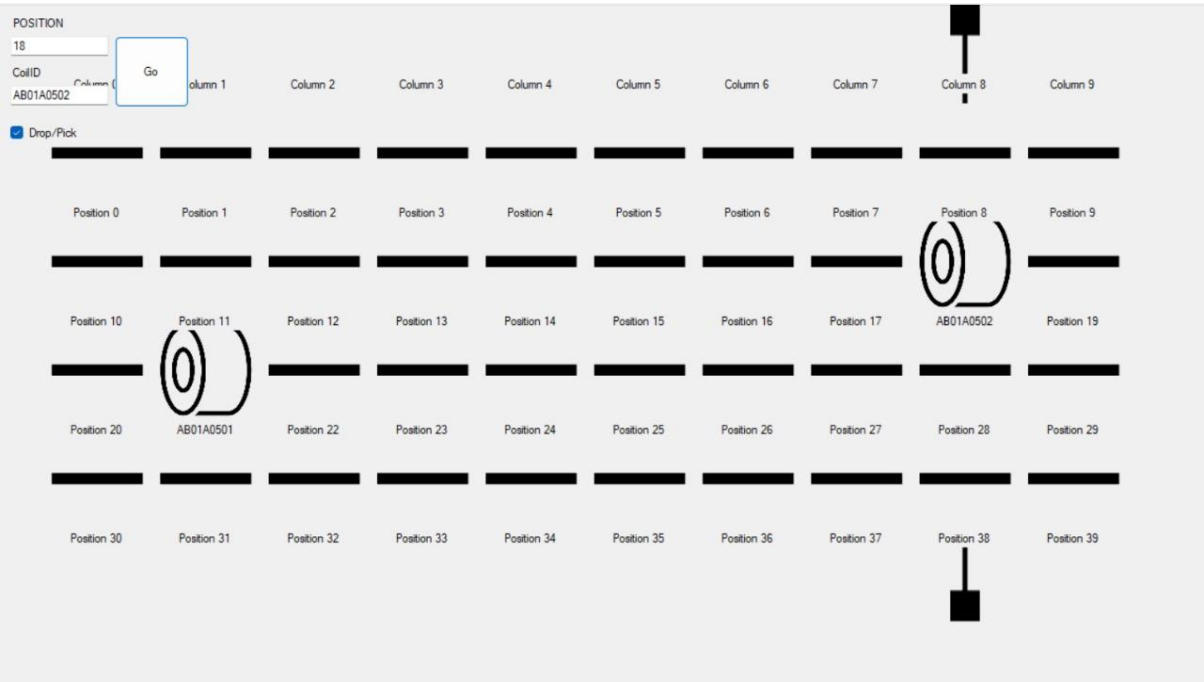
```
          <xsd:element name="resheader">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" msdata:Ordinal="1" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required" />
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
  <resheader name="resmimetype">
    <value>text/microsoft-resx</value>
  </resheader>
  <resheader name="version">
    <value>2.0</value>
  </resheader>
  <resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <resheader name="writer">
    <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <assembly alias="System.Windows.Forms" name="System.Windows.Forms, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" />
  <data name="pngegg" type="System.Resources.ResXFileRef, System.Windows.Forms">
    <value>..\Resources\pngegg.png;System.Drawing.Bitmap, System.Drawing, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a</value>
  </data>
  <data name="Coil" type="System.Resources.ResXFileRef, System.Windows.Forms">
    <value>..\Resources\Coil.png;System.Drawing.Bitmap, System.Drawing, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a</value>
  </data>
  <data name="Cut_back" type="System.Resources.ResXFileRef, System.Windows.Forms">
    <value>..\Resources\Cut_back.png;System.Drawing.Bitmap, System.Drawing, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a</value>
  </data>

</root>
```

SAMPLE INPUT

OUTPUT:

OUTPUT:


POSITION
18
CoilID
AB01A0502
Go
Drop/Pick
Column 1
Column 2
Column 3
Column 4
Column 5
Column 6
Column 7
Column 8
Column 9
Position 0
Position 1
Position 2
Position 3
Position 4
Position 5
Position 6
Position 7
Position 8
Position 9
Position 10
Position 11
Position 12
Position 13
Position 14
Position 15
Position 16
Position 17
AB01A0502
Position 19
Position 20
AB01A0501
Position 22
Position 23
Position 24
Position 25
Position 26
Position 27
Position 28
Position 29
Position 30
Position 31
Position 32
Position 33
Position 34
Position 35
Position 36
Position 37
Position 38
Position 39

CONCLUSION

In this project C# and .NET are mixed. Visual studio makes me to easy this project. C# “the right way” requires a lot of typing. Implementing interfaces even if visual studio has a feature to automatically add the required functions, feel like I am repeating the same thing a thousand times.
C# shares many of the features of other object-oriented languages such as C++. C# includes supports for virtual methods, abstract classes, and method overloading. C# offers an additional feature that promotes more reusable and more robust classes.
The documentation provided covers the key aspects of the Coil movement, including its features, code structure, and customization options. It also delves into the broader context of Yard Management Systems, their uses in companies, and the benefits they offer in terms of inventory management, yard optimization, scheduling and coordination, visibility and reporting, automation and efficiency, compliance and safety, and integration with other enterprise systems.

By implementing a comprehensive Yard Management System, companies can streamline their yard operations, reduce costs, and enhance customer service by ensuring the timely and accurate movement of goods, materials, and equipment. The Coil Movement App serves as a starting point for understanding the core functionality of a YMS and can be further expanded upon to create more complex and feature-rich applications tailored to specific industry needs.

As technology continues to advance, Yard Management Systems will play an increasingly important role in optimizing supply chain operations and driving business success in the manufacturing, warehousing, and transportation industries.
Overall, I had learned many things while working on this project. It had been a pleasure while working in this project.

THANK YOU

Thank you, MR. Manish Upadhayay, Dy. Head, Automation, Tata BlueScope Steel Pvt. Ltd. who helped us a lot in completing this project &giving wonderful opportunity to complete my project training in this big organization. Heartfelt thanks to MR. Abhishek Singh who helped us a lot in learning new languages & building up this project. I would like to express my sincere gratitude for your invaluable guidance and support throughout the development of the project. Your expertise and insights have been instrumental in shaping this application and helping me understand the intricacies of Yard Management Systems.

From the initial conceptualization to the final implementation, your mentorship has been truly invaluable. Your deep knowledge of supply chain management, logistics, and material handling operations has been crucial in ensuring that the Coil Movement App accurately reflects the real-world challenges and solutions in this domain. Your patience in explaining the various components of a Yard Management System, such as inventory tracking, crane movement, and yard optimization, has greatly enhanced my understanding of this domain. The detailed documentation you provided, covering the features, code structure, and customization options, has been an invaluable resource that will continue to guide me in further developing and enhancing the application.

I am particularly grateful for your guidance in integrating the with the broader context of Yard Management Systems and their applications in companies like TATA BLUESCOPE STEEL. This holistic perspective has helped me appreciate the importance of such systems in optimizing supply chain operations and driving business success. Your unwavering support and constructive feedback throughout the development process have been instrumental in shaping the Coil Movement App into a robust and meaningful application. I am truly

grateful for the time and effort you have invested in mentoring me and helping me grow as a developer.

Thank you for your dedication, patience, and invaluable expertise. I am confident that the knowledge and skills I have gained through this project will serve me well in my future endeavours. I look forward to continuing to learn from you and applying these lessons to create even more impactful solutions.

Lastly, Thanks to all members of TATA BLUESCOPE STEEL who directly or indirectly helped me in completing this project successfully.

Sincerely,

ANANT SHRIVASTAVA