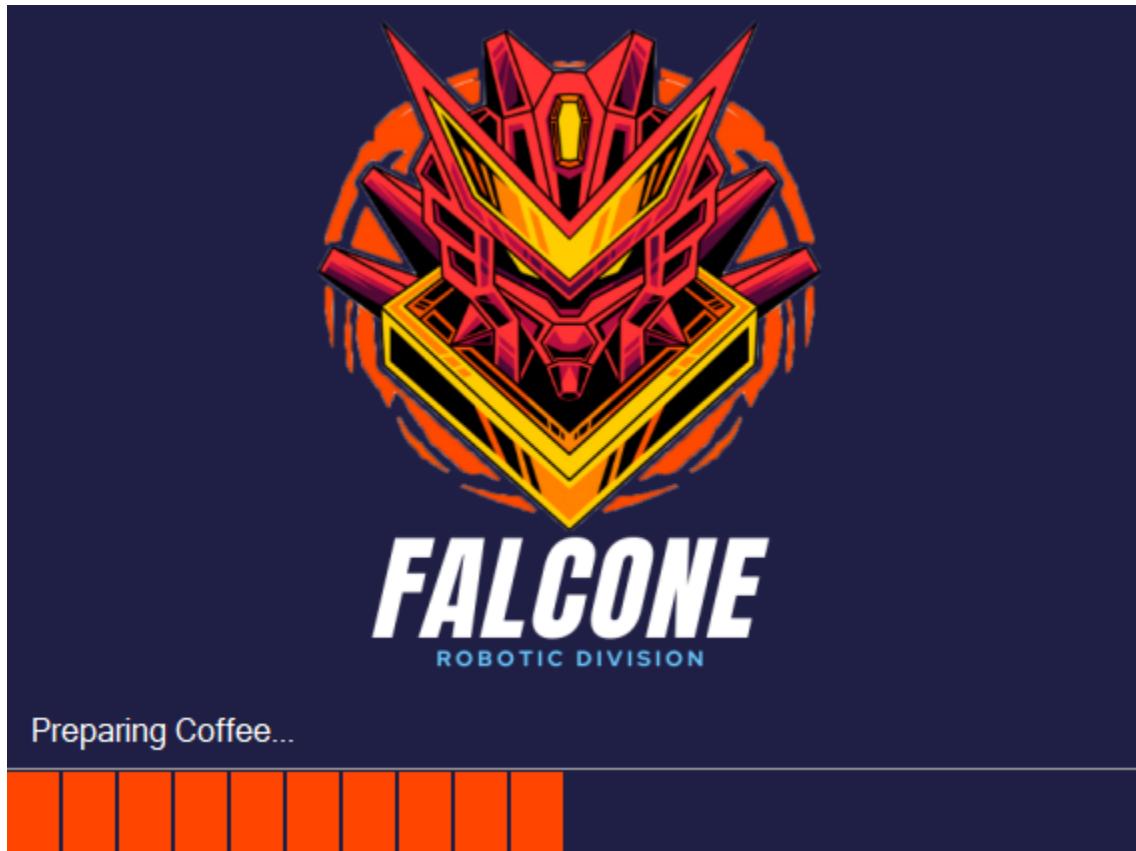


TP3 Final - Laboratorio 2

Amilcar Facundo Falcone - 2D.

Fabrica de Robots

9 de Julio del 2021

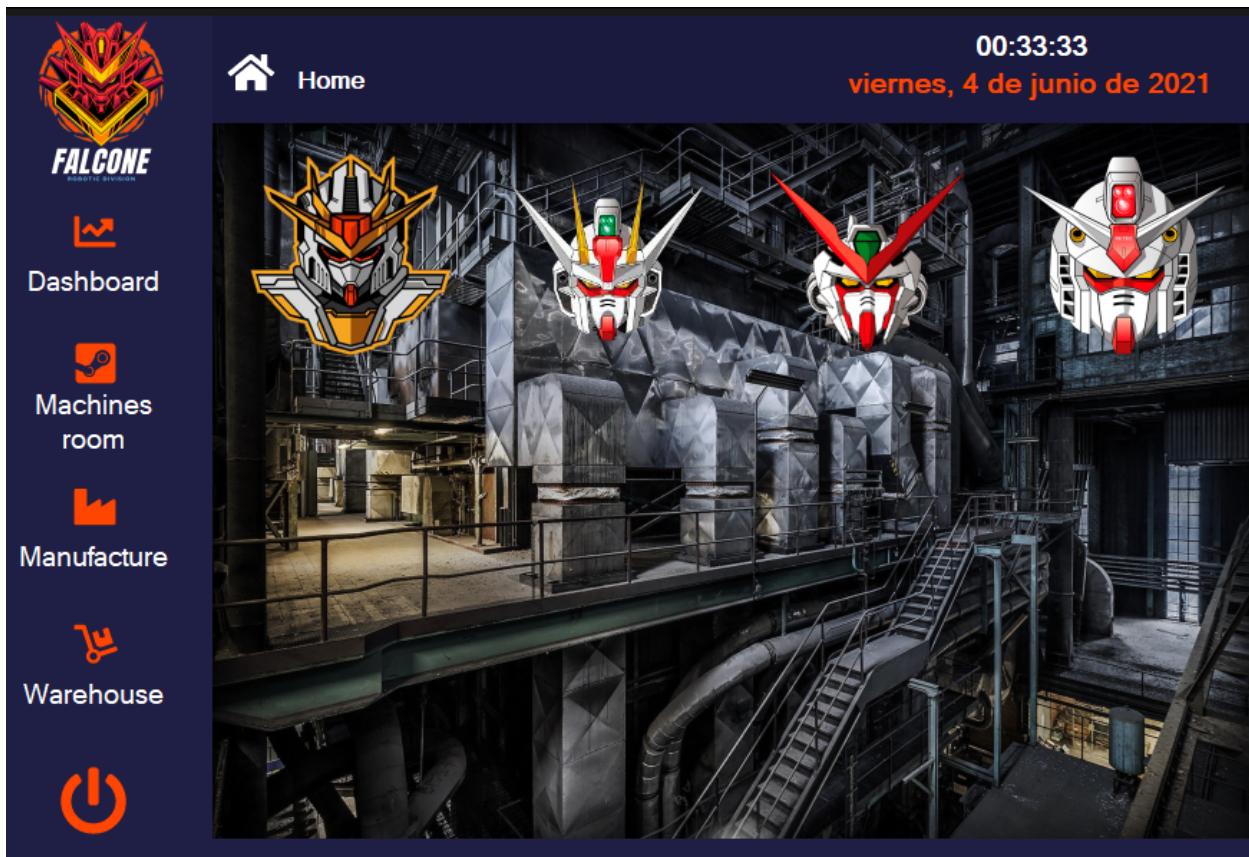


Explicación de la mini app

El siguiente proyecto simula una pequeña fábrica de robots en la que se podrán crear algunos diseños propuestos. Dicho programa para que funcione correctamente deberá inicializar primero su stock de materiales y de ser correcta su cantidad, con las que necesita el modelo robótico, procederá a fabricarlo para luego poner el robot en su almacén. En el almacén o warehouse, se podrá ver los modelos fabricados como así también el stock actual de materiales.

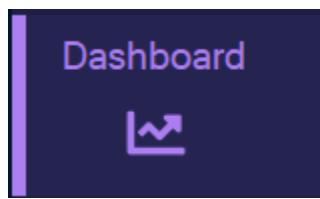
Puedes verlo funcionando en: [Pequeño Tour Por el TP3](#)

Pequeño Lobby

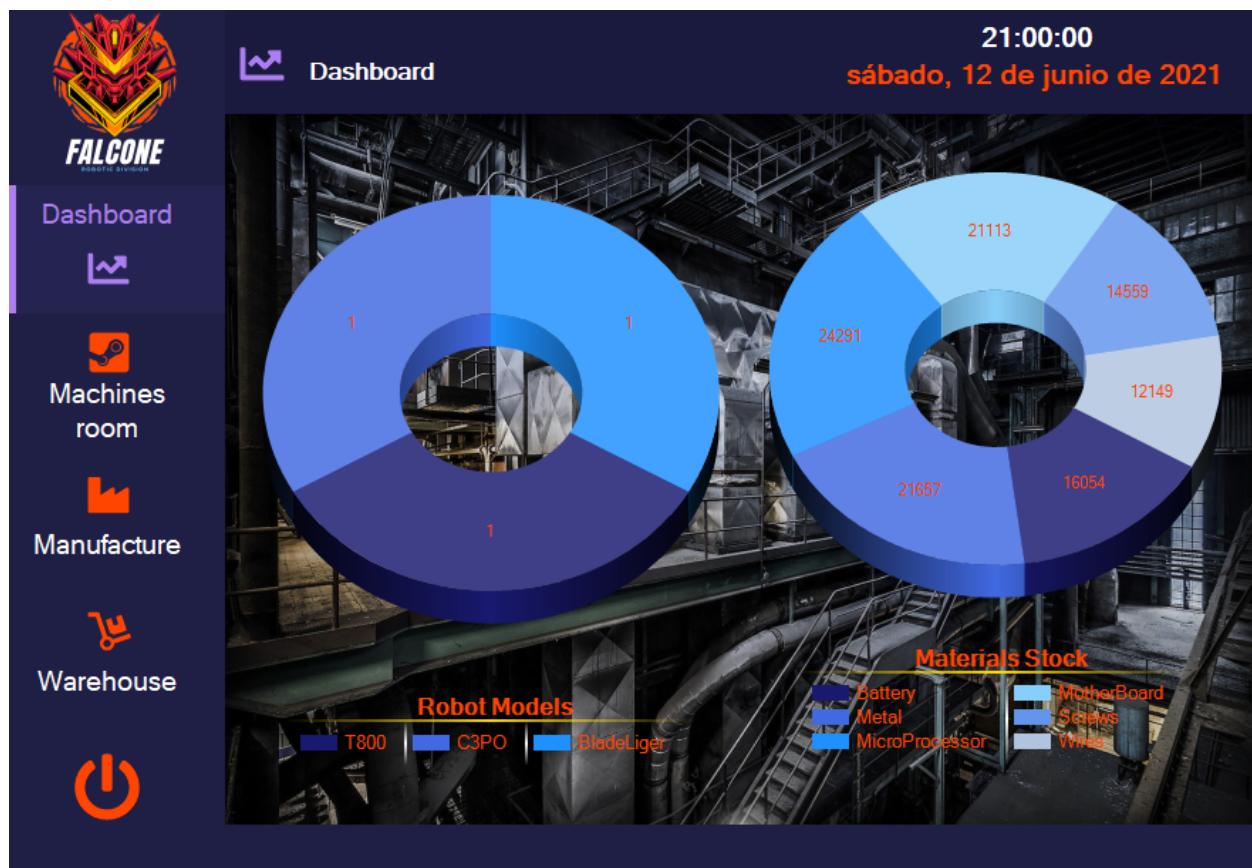


Descripción:

El programa te da la posibilidad de crear stock tanto de materiales como de robots, al consumir esos materiales.



En el botón 'Dashboard' podrán ver dos gráficos de dona con la cantidad y porcentaje de materiales y robots que hay en el stock de la fábrica, en caso de no haber robots y/o stock de materiales, no se visualizarán los gráficos.





Si es la primera vez que ejecutas el programa, Empezá por acá

En el botón '**Machines Room**' podrá inicializar la fábrica e importar materiales a la misma usando números random para aumentar el stock. La primera vez que ejecutes el programa, Acá aparecerán dos botones, uno de ellos será para inicializar la fábrica, al darle clic el botón desaparecerá para nunca más volver. Con el otro botón se podrá importar materia prima a la fábrica (se generan de manera random) y los valores se podrán visualizar en el gráfico de dona. Cada vez que se apriete el botón de '**Import Stock**', los valores del gráfico se actualizarán. Al iniciar el programa nuevamente (Con stock generado), El botón '**Initialize Factory**' no estará visible, ya que esta acción solo se puede hacer una vez. Se cargará un archivo serializado en el que sus registros serán cargados en la memoria del programa.

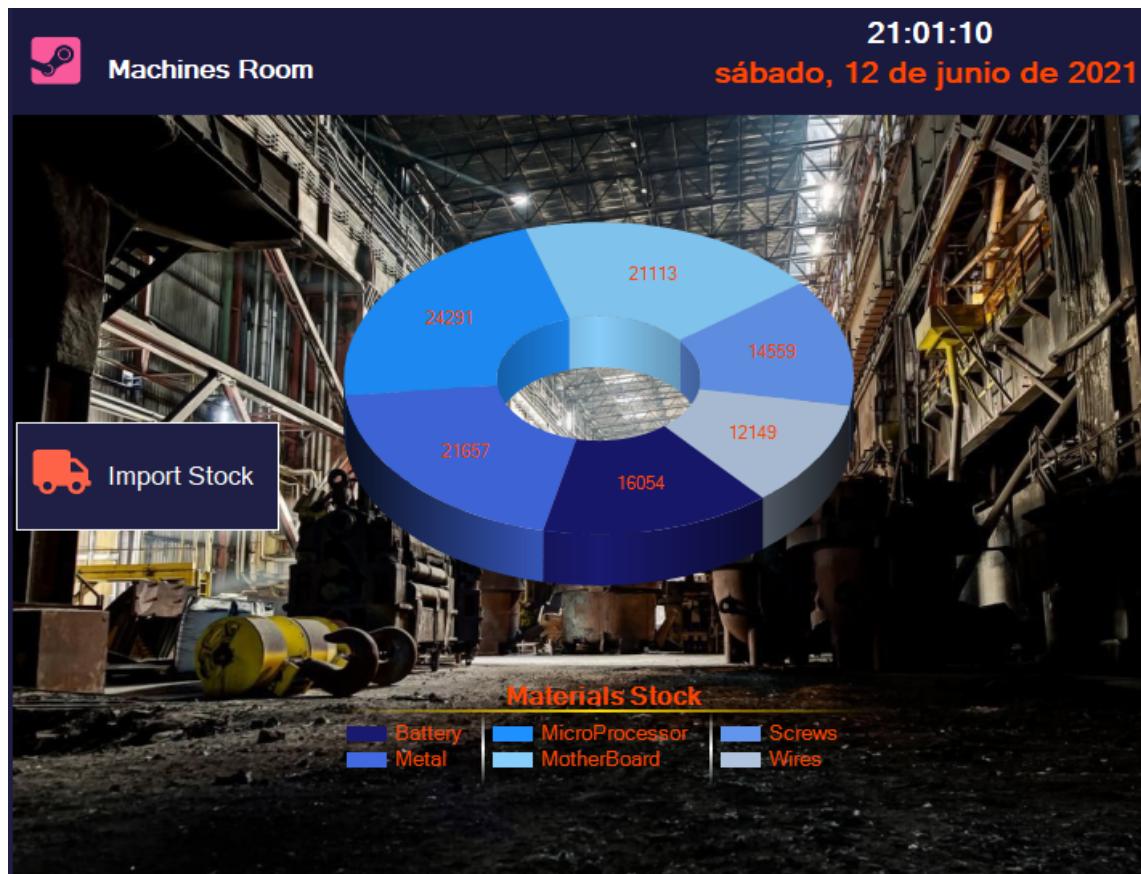


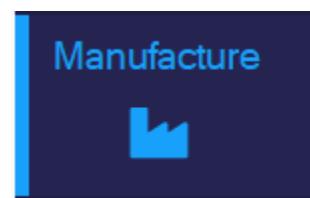
Machines Room

21:19:48

sábado, 12 de junio de 2021



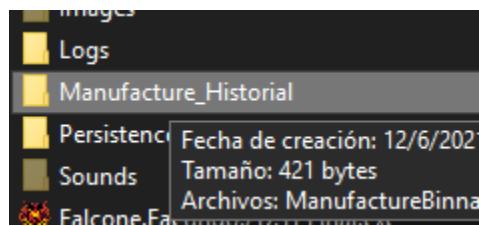




Manufacture: En el botón '**Manufacture**' podrán crear hasta 9 tipos distintos de robots (en caso de que la cantidad de materiales del almacén sean suficientes, caso contrario tirara una excepción mediante un formulario).



Por cada robot fabricado, se generará o re escribirá (en caso de existir) un archivo con la información del robot creado en el directorio
root\Manufacture_Historial\ManufactureBinnacle.txt



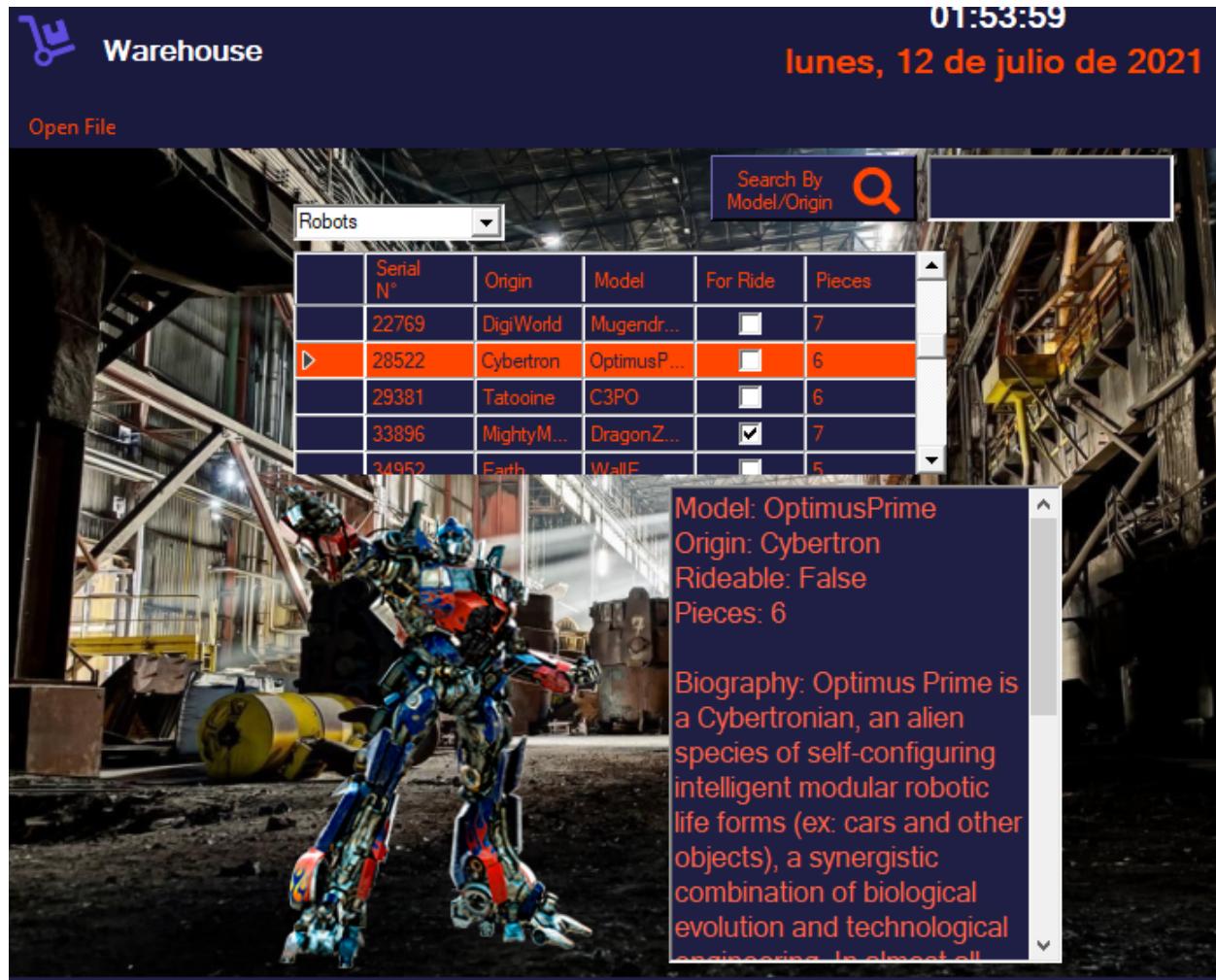
```
12/6/2021 23:30:09 - Product: T800
Manufacture Batch: 510-702-627
Manufacture Date: 12/6/2021 23:30:09
```

```
12/6/2021 23:30:17 - Product: C3PO
Manufacture Batch: 548-789-873
Manufacture Date: 12/6/2021 23:30:17
```

```
12/6/2021 23:30:25 - Product: WallE
Manufacture Batch: 509-838-852
Manufacture Date: 12/6/2021 23:30:25
```



En el botón 'Warehouse' se podrá ver el stock de la fábrica, tanto de materiales como de robots, como así también una breve información sobre cada robot en el almacén.



The screenshot shows a software interface titled "Warehouse". At the top right, it displays the time "01:53:59" and the date "lunes, 12 de julio de 2021". On the left, there is a "Open File" button. In the center, there is a search bar with the placeholder "Search By Model/Origin" and a magnifying glass icon. A dropdown menu labeled "Robots" is open, showing a table with the following data:

	Serial N°	Origin	Model	For Ride	Pieces
	22769	DigiWorld	Mugendr...	<input type="checkbox"/>	7
▶	28522	Cybertron	OptimusP...	<input type="checkbox"/>	6
	29381	Tatooine	C3PO	<input type="checkbox"/>	6
	33896	MightyM...	DragonZ...	<input checked="" type="checkbox"/>	7
	34952	Earth	WallE	<input type="checkbox"/>	5

A large image of Optimus Prime stands prominently in the background of the interface. A tooltip window is open over the Optimus Prime entry in the table, displaying the following information:

Model: OptimusPrime
Origin: Cybertron
Rideable: False
Pieces: 6

Biography: Optimus Prime is a Cybertronian, an alien species of self-configuring intelligent modular robotic life forms (ex: cars and other objects), a synergistic combination of biological evolution and technological engineering. In almost all

Temas Utilizados para el programa:

- *Excepciones*
- *Test Unitarios*
- *Tipos Genericos*
 - *Interfaces*
 - *Archivos*
 - *Serialización*
- *Métodos de Extensión*
 - *Delegados*
 - *Eventos*
 - *Hilos*
 - *SQL*

Excepciones:

Implementado en varias secciones de los formularios para controlar posibles incidentes, por ejemplo una de las implementadas es:

```

14 referencias
public class InsufficientMaterialsException : Exception {

    #region Builders

        /// <summary>
        /// Creates an exception with the message.
        /// </summary>
        /// <param name="message">Message to show in the exception.</param>
        2 referencias
    public InsufficientMaterialsException(string message) : this(message, null) { }

        /// <summary>
        /// Creates an exception with the message and its innerException.
        /// </summary>
        /// <param name="message">Message to show in the exception.</param>
        /// <param name="innerException">InnerException of the exception.</param>
        1 referencia
    public InsufficientMaterialsException(string message, Exception innerException) : base(

```

La cual será lanzada al querer fabricar un robot sin tener la cantidad necesaria de materiales en:
TP4Final.GUI/TP4Final.GUI.Factory/TP4Final.GUI.Factory.Manufacture/frmManufacture/ConfigureRobotForBuild()

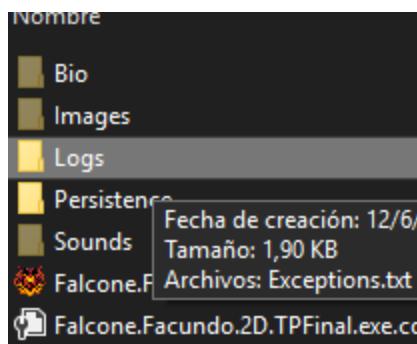
```

prototype = RobotFactory.CreateMultiplePiecesAmdAddToStock(metalType, origin, modelName, amountOfMaterials, am
filename = prototype.Model.ToString();
absBioPath = $"{pathBiography}{filename}.bin";
prototype.LoadBioFile(absBioPath);

if (RobotFactory.QualityControl(prototype, amountPieces)) {
    if (RobotFactory.AddRobotToWarehouse(prototype)) {
        MyPlayer.Play($"BuildingForm", false);
        frmLobby.FormShowDialogHandler(new frmBuilding());
        RobotFactory.SaveDataOfFactory();
        MyPlayer.Play($"Create{prototype.Model}", false);
        frmLobby.FormShowDialogHandler(new frmISOCertified(prototype.Model.ToString()));
    }
} else {
    if (RobotFactory.DissasembleRobot(prototype)) {
        throw new QualityControlFailedException("There have a difference in the amount of pieces, Quality control failed");
    }
}
prototype = null;
} else {
    throw new InsufficientMaterialsException("Insufficients Materials to build this model of robot, you need to import more materials");
}

```

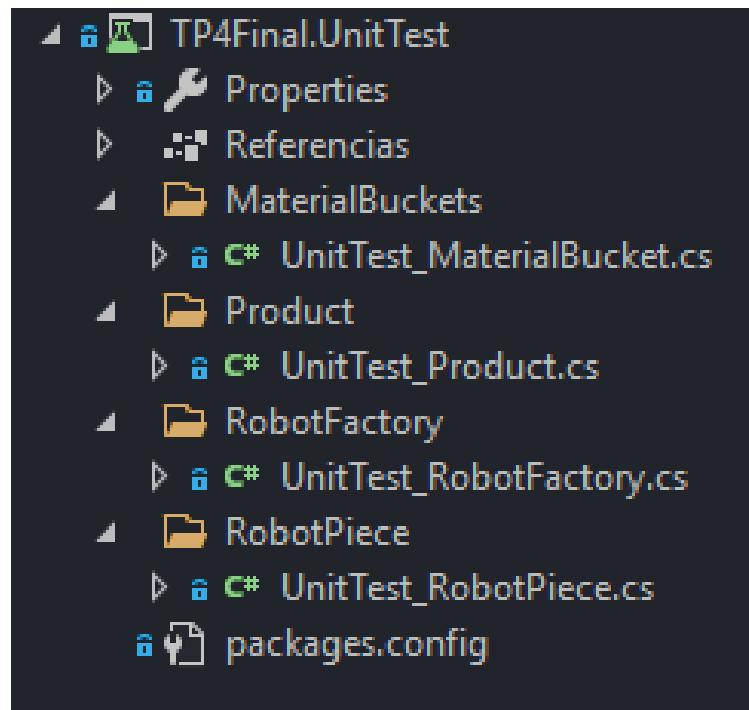
Por cada excepción lanzada, se creara o re escribirá (en caso de existir) un archivo con la información de la excepción en el directorio root\Logs\Exceptions.txt



```
12/6/2021 23:10:27 - System.NullReferenceException: Referencia a objeto no establecida como instancia de un objeto.  
  en FactoryForms.frmWarehouse.dgvRobots_CellContentClick(Object sender, DataGridViewEventArgs e) en E:\Git_UTN\GitHub\tp_laboratorio_2\TP_03_FacundoFalcone\Falcone.Facundo.2D.TPFinal\Factory\Warehouse\frmWarehouse.cs: línea 110  
-----  
12/6/2021 23:19:00 - System.NullReferenceException: Referencia a objeto no establecida como instancia de un objeto.  
  en FactoryForms.frmWarehouse.dgvRobots_CellContentClick(Object sender, DataGridViewEventArgs e) en E:\Git_UTN\GitHub\tp_laboratorio_2\TP_03_FacundoFalcone\Falcone.Facundo.2D.TPFinal\Factory\Warehouse\frmWarehouse.cs: línea 110  
-----  
12/6/2021 23:22:39 - System.NullReferenceException: Referencia a objeto no establecida como instancia de un objeto.  
  en FactoryForms.frmWarehouse.dgvRobots_CellContentClick(Object sender, DataGridViewEventArgs e) en E:\Git_UTN\GitHub\tp_laboratorio_2\TP_03_FacundoFalcone\Falcone.Facundo.2D.TPFinal\Factory\Warehouse\frmWarehouse.cs: línea 110  
-----
```

Test Unitarios:

En el proyecto 'UnitTestProjectTPFinal' hay test unitarios para la mayoría de las clases, donde se testean métodos que hagan lo correspondiente.



Tipos Genericos:

Implementados en interfaces por un tema de practicidad, se podrán encontrar en el proyecto
bibliotecas de clases

'TP4Final.Models/TP4Final.Models.Classes/TP4Final.Models.Classes.Interfaces'

```
namespace Interfaces {
    2 referencias
    public interface IFilesManager<T> {
        6 referencias
        bool Save(string path, string filename, T dataToSave);
        4 referencias
        T Read(string filePath);
    }
}
```

Implementación en clase 'SerialManager<T>'.

```
8 referencias
public class SerialManager<T> : IFilesManager<T> {

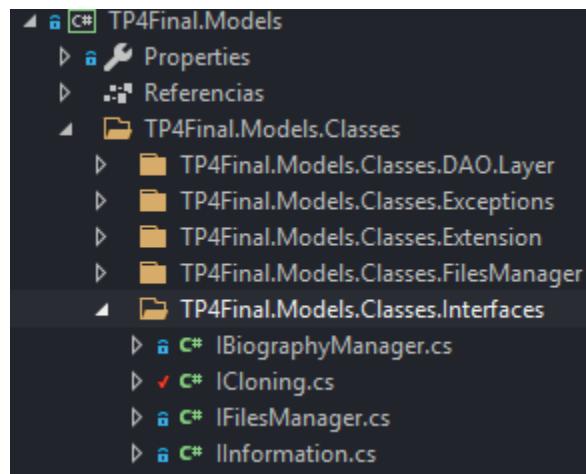
    #region Methods

    /// <summary>
    /// Reads the data from the path passed by parameter.
    /// </summary>
    /// <param name="filePath">Path to read the file.</param>
    /// <returns>True if can read the file, otherwise returns false.</returns>
    4 referencias
    public T Read(string filePath) {
        T aux;
        try {
            using (XmlTextReader reader = new XmlTextReader(filePath)) {
                XmlSerializer serial = new XmlSerializer(typeof(T));
                aux = (T)serial.Deserialize(reader);
            }
        } catch (Exception ex) {
            throw new Exception("Something get wrong trying reading Buckets", ex);
        }

        return aux;
    }
}
```

Interfaces:

se podrán encontrar en el proyecto bibliotecas de clases
'TP4Final.Models/TP4Final.Models.Classes/TP4Final.Models.Classes.Interfaces'



La gran mayoría de las interfaces son genéricas, por ejemplo:

```
1 referencia
public interface ICloning<T>
    where T : RobotPiece {

    /// <summary>
    /// Clones the list passed by parameter and returns the clone.
    /// </summary>
    /// <param name="listToClone">List to clone.</param>
    /// <returns>The list clonned.</returns>
    2 referencias
    List<T> CloneList(List<T> listToClone);

}
```

Implementadas en clases como 'Robot', interfaces con métodos que nada tienen que ver con la clase Robot pero que son útiles usarlos ya que proveen de funcionalidades extras que hacen a la clase.

```
43 referencias
public class Robot : IInformation, ICloning<RobotPiece>, IBiographyManager {

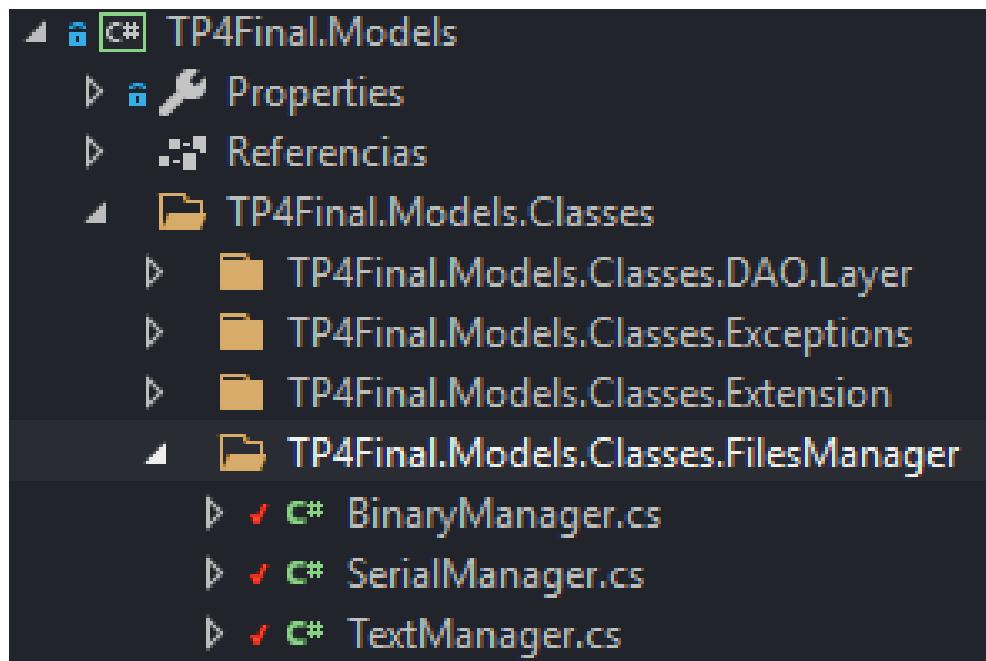
    #region Attributes

    private int serialNumber;
    private EOrigin origin;
    private EModelName modelName;
    private bool isRideable;
    private string biography;
    private List<RobotPiece> pieces;

    #endregion
```

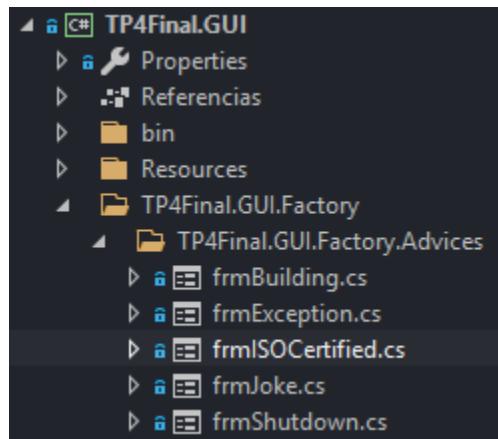
Archivos:

Usados para escribir en txt el historial de fabricación de robots, como así también los 'logs' de las excepciones. Se implementa una interfaz en una clase de administrador de archivos



```
/// <summary>
/// Saves a stream into a file.
/// </summary>
/// <param name="path">Path to save the file.</param>
/// <param name="fileName">Name of the file to save in the path.</param>
/// <param name="dataToSave">Data to write into the file.</param>
/// <returns>True if can write the file, otherwise returns false.</returns>
6 referencias
public bool Save(string path, string filename, string dataToSave) {
    string absPath = $"{path}\\\{filename}";
    StringBuilder data = new StringBuilder();
    data.AppendLine($"{DateTime.Now} - {dataToSave}\n-----");
    try {
        if (!Directory.Exists(path)) {
            Directory.CreateDirectory(path);
        }
        using (StreamWriter sw = File.AppendText($"{absPath}"))
        {
            sw.WriteLine(data);
            return true;
        }
    } catch (Exception ex) {
        throw new Exception("Something get wrong trying saving Document", ex);
    }
}
```

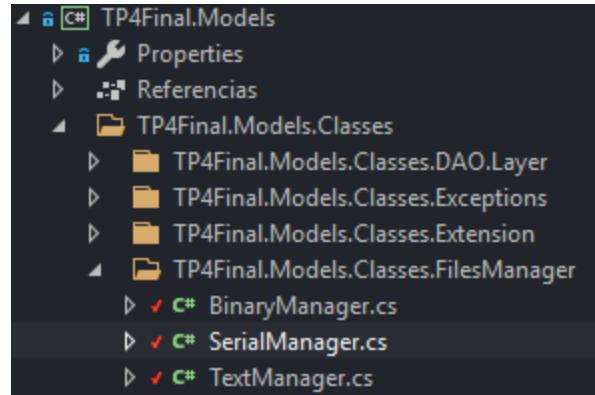
Usados en formularios al realizar ciertas acciones.



```
/// <summary>
/// EventHandler of the Load.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 referencia
private void frmISOCertified_Load(object sender, EventArgs e) {
    this.Opacity = 0.0;
    this.pbCertified.Value = 0;
    this.pbCertified.Minimum = 0;
    this.pbCertified.Maximum = 100;
    this.timeFadeIn.Start();
    ipbImage.ImageLayout = ImageLayout.Zoom;
    ipbImage.BackgroundImage = Image.FromFile($"{this.systemImagePath}\\" + productName + ".png");
    warrantyMessage = WarrantyMessage();
    rtbWarrantyMessage.Text = warrantyMessage;
    logger.SaveFull(path, filename, warrantyMessage);
}
```

Serialización:

Usados en Interfaces e implementado en la clase SerialManager, del tipo genérica, para serializar y deserializar archivos y listas.

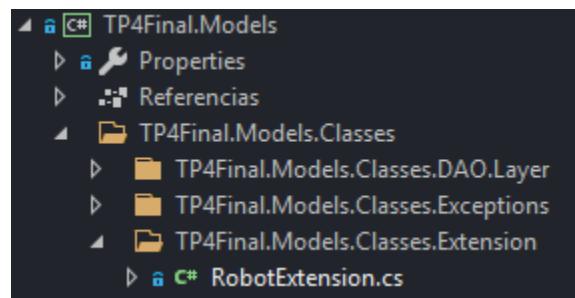


```
/// <summary>
/// Reads the data from the path passed by parameter.
/// </summary>
/// <param name="filePath">Path to read the file.</param>
/// <returns>True if can read the file, otherwise returns false.</returns>
4 referencias
public T Read(string filePath) {
    T aux;
    try {
        using (XmlTextReader reader = new XmlTextReader(filePath)) {
            XmlSerializer serial = new XmlSerializer(typeof(T));
            aux = (T)serial.Deserialize(reader);
        }
    } catch (Exception ex) {
        throw new Exception("Something get wrong trying reading Buckets", ex);
    }

    return aux;
}
```

Metodos de Extension:

Hice dos métodos de extensión de la clase Robot, ubicados en la siguiente ruta



Los cuales se encargan de formatear las ID de las piezas robóticas basadas en el serial del robot, como tambien de formatear los ID de los materiales de cada pieza robótica, basado en el ID de cada pieza.

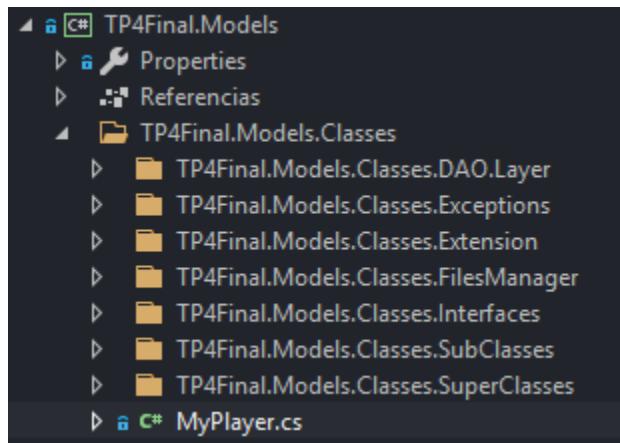
```
1 referencia
public static void FormatRobotPieceID(this RobotPiece piece, int robotSerial, int indexPiece) {
    string prefix = string.Empty;
    switch (piece.PieceType) {
        case EPieceType.Head:
            prefix = "HE";
            break;
        case EPieceType.Torso:
            prefix = "TO";
            break;
        case EPieceType.UpperLimb:
            prefix = "UL";
            break;
        case EPieceType.LowerLimb:
            prefix = "LL";
            break;
        case EPieceType.Tail:
            prefix = "TA";
            break;
    }
    piece.PieceID = $"{robotSerial}-{prefix}-{indexPiece}";
}
```

```
/// <summary>
/// Extends the functionality of a Robot, this method was made for set
/// the serial number of the robot into its pieces and the material of
/// each robot piece.
/// </summary>
/// <param name="thisRobot">Class to extend.</param>
/// <param name="aRobot">Entity to modify data.</param>
1 referencia
public static void SetSerialNumber(this Robot thisRobot) {
    foreach (RobotPiece item in thisRobot.RobotPieces) {
        item.AssociatedRobotSerial = thisRobot.SerialNumber;
        item.FormatRobotPieceID(item.AssociatedRobotSerial, thisRobot.RobotPieces.IndexOf(item));
        foreach (MaterialBucket itemBucket in item.RawMaterial) {
            itemBucket.AssociatedRobotSerial = thisRobot.SerialNumber;
            itemBucket.AssociatedPieceID = item.PieceID;
        }
    }
}
```

Delegados, Eventos e Hilos:

Estos 3 temas se aplicaron juntos para manejar audios, como así imágenes del tipo gif.

Por ejemplo, en la clase MyPlayer ubicada en



Tiene un evento y delegado declarado

```
public delegate void SoundPlayerHandler(object songName);

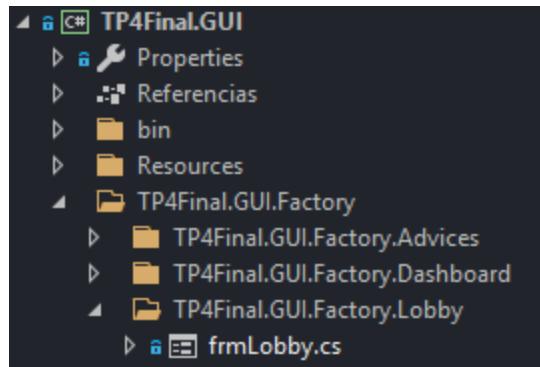
17 referencias
public class MyPlayer {

    #region Attributes

    public event SoundPlayerHandler ESoundPlayer;
    private WindowsMediaPlayer player;
    private bool isPlaying;
    private bool isLooping;
    private string format = ".mp3";

    #endregion
```

el cual el form lobby



Lo usa de la siguiente manera

```
/// <summary>
/// Creates a soundPlayer and plays the sound.
/// </summary>
/// <param name="soundName">Name of the sound to play.</param>
3 referencias
private void MyPlayerMainMusic(object soundName) {
    if (this.InvokeRequired) {
        SoundPlayerHandler sp = new SoundPlayerHandler(MyPlayerMainMusic);
        this.BeginInvoke(sp, new object[] { (string)soundName});
    } else {
        MyPlayer player = new MyPlayer();
        player.Play((string)soundName);
    }
}
```

Creando un hilo cada vez que se utiliza el método y agregándolo a una lista de hilos.

```
/// <summary>
/// Plays a sound in another thread.
/// </summary>
/// <param name="musicName">Name of the sound.</param>
5 referencias
private void PlayMusic(string musicName) {
    Thread playerThread = new Thread(new ParameterizedThreadStart(this.MyPlayerMainMusic));
    playerThread.Start(musicName);
    threads.Add(playerThread);
}
```

Se asocia en el constructor

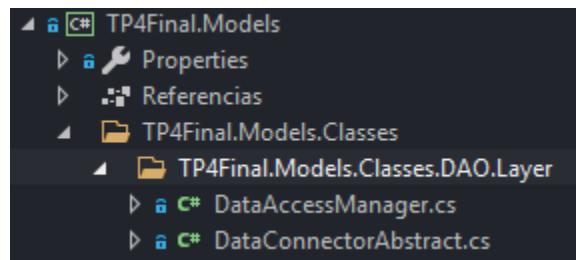
```
5 referencias
public frmLobby() {
    InitializeComponent();
    myDelPlayer = new MyPlayer();
    myDelPlayer.ESoundPlayer += MyPlayerMainMusic;
    activeForm = null;
```

y desasociandolo en el formClosing, como así tambien se terminan todos los hilos que estuviesen activos en la lista.

```
private void frmLobby_FormClosing(object sender, FormClosingEventArgs e) {
    if (MessageBox.Show("Are you sure to quit?", "Leaving", MessageBoxButtons.YesNo) == DialogResult.Yes) {
        RobotFactory.SaveDataOfFactory();
        formShut = new frmShutdown();
        formShut.ShowDialog();
        myDelPlayer.ESoundPlayer -= MyPlayerMainMusic;
        this.KillThreads();
        this.Dispose();
    } else {
        e.Cancel = true;
    }
}
```

```
    /// <summary>
    /// Kills all possibly alive threads.
    /// </summary>
    1 referencia
    private void KillThreads() {
        foreach (Thread item in threads) {
            if (item.IsAlive) {
                item.Abort();
            }
        }
    }
```

SQL:



Se creó una clase abstracta 'DataConnectorAbstract' de la cual hereda 'DataAccessManager'. 'DataConnectorAbstract' Tiene las configuraciones básicas para poder conectar con la DB, mientras que 'DataAccessManager' Tiene dentro métodos usados para las diferentes consultas que se harán en la DB.

```
0 referencias
public abstract class DataConnectorAbstract {

    #region Attributes

    private static string connString;
    private static SqlConnection myConnection;
    private static SqlCommand myCommand;
    private static string server = "localhost";
    private static string DataBase = "TPFinal";
    private static bool tConection = true;

    #endregion

    #region Builders

    0 referencias
    static DataConnectorAbstract() {
        connString = $"Server = {server} ; Database = {DataBase}; Trusted_Connection = {tConection}; ";
        myConnection = new SqlConnection(connString);
        myCommand = new SqlCommand();
        myCommand.Connection = myConnection;
        myCommand.CommandType = CommandType.Text;
    }
}
```

```
43 referencias
public class DataAccessManager : DataConnectorAbstract {

    #region Methods

    /// <summary>
    /// Converts a DataRow into a Robot Object.
    /// </summary>
    /// <param name="item">DataRow To convert into a Robot.</param>
    /// <param name="pieces">List of pieces of the robot.</param>
    /// <returns>An instance of Robot Type.</returns>
    1 referencia
    private static Robot DataRowToRobot(DataRow item, int robotSerial, List<RobotPiece> pieces) {
        Robot myRobot;
        Enum.TryParse(item["Origin"].ToString(), out EOrigin origin);
        Enum.TryParse(item["ModelName"].ToString(), out EmodelName model);
        bool rideable = Convert.ToBoolean(item["IsRideable"]);
        myRobot = new Robot(origin, model, pieces, rideable);
        myRobot.SerialNumber = robotSerial;

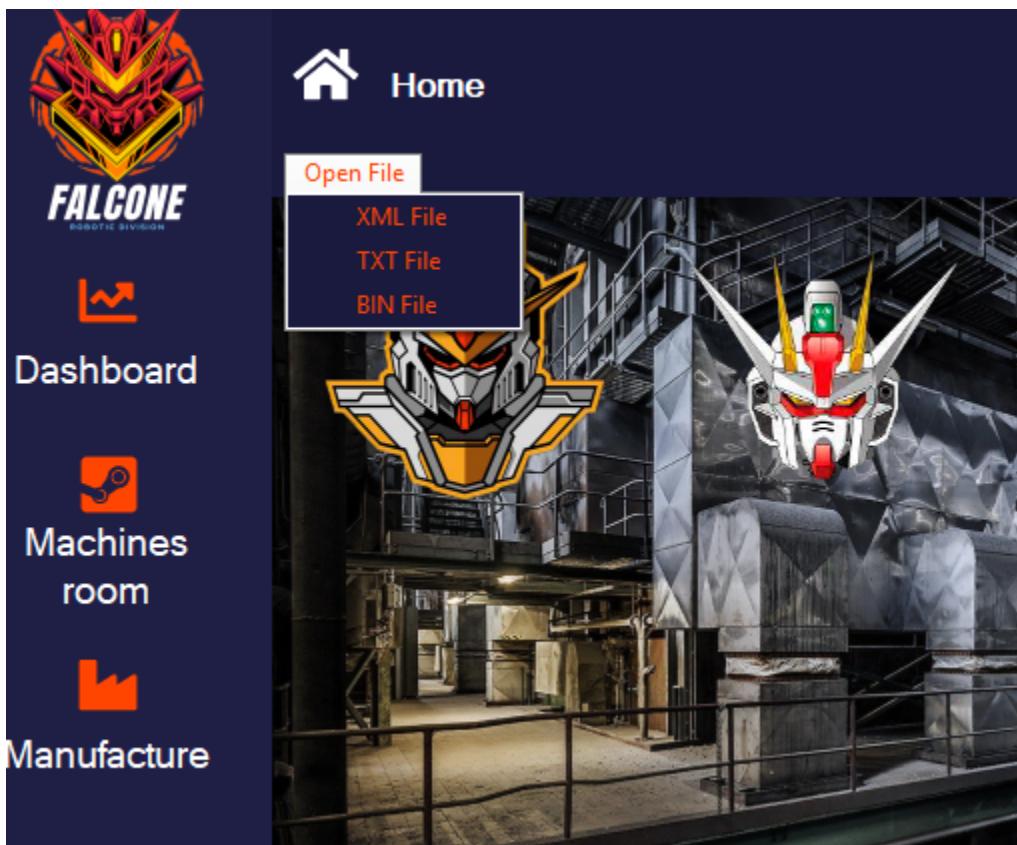
        return myRobot;
    }
}
```

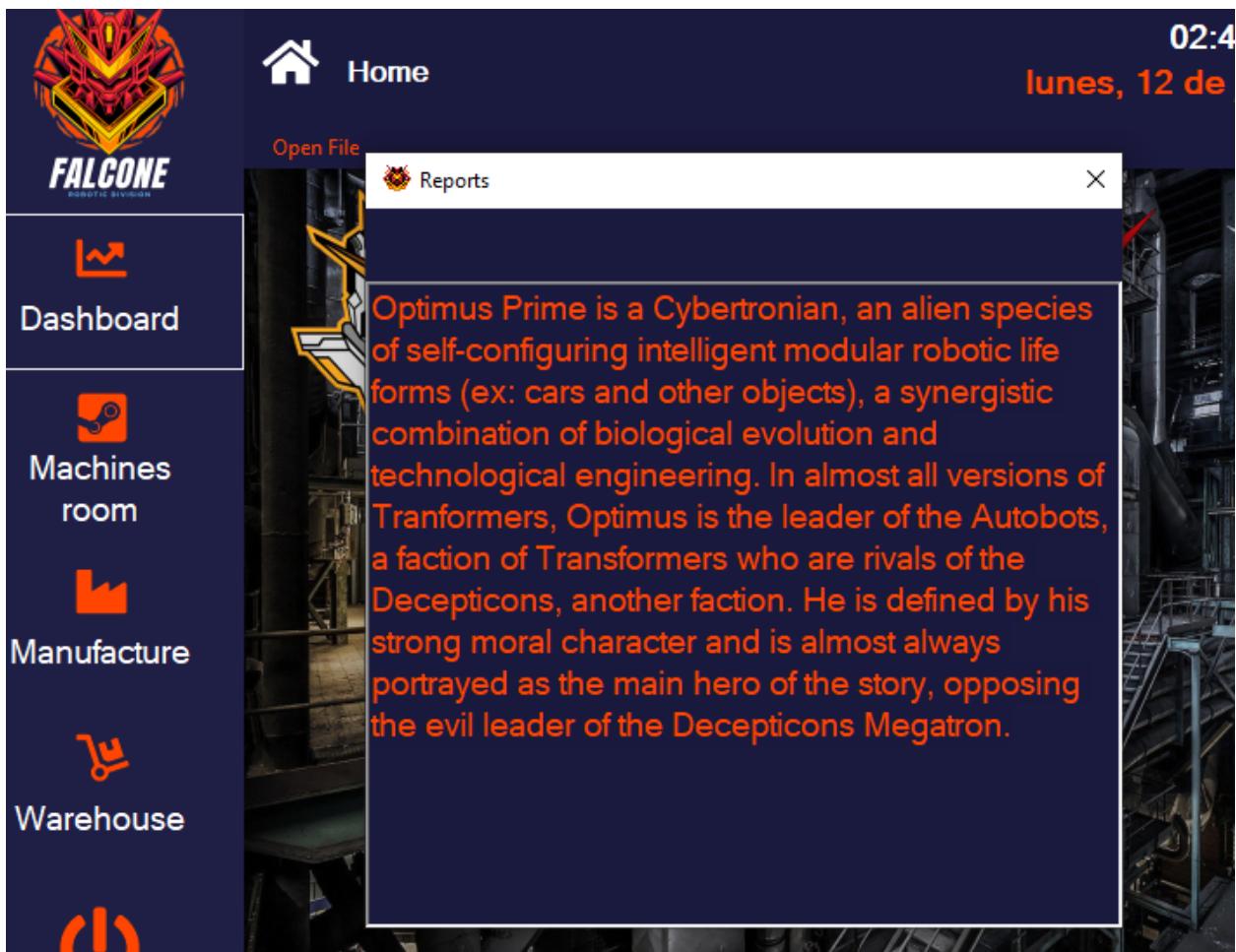
```
/// <summary>
/// Gets the list of robots.
/// </summary>
/// <returns>The list of all the robots in the warehouse.m</returns>
1 referencia
public static List<Robot> GetRobots() { //TODO: Fix This
    List<Robot> robots = new List<Robot>();
    List<RobotPiece> pieces;
    Robot actualRobot;
    try {
        DataAccessManager.MyCommand.CommandText = "Select * from Robots";
        DataAccessManager.MyConection.Open();
        using (SqlDataReader myReader = DataAccessManager.MyCommand.ExecuteReader()) {
            DataTable myDT = new DataTable();
            myDT.Load(myReader);
            foreach (DataRow item in myDT.Rows) {
                int robotSerial = Convert.ToInt32(item["SerialNumber"]);
                pieces = DataAccessManager.GetPieces(robotSerial);
                actualRobot = DataAccessManager.DataRowToRobot(item, robotSerial, pieces);
                robots.Add(actualRobot);
            }
        }
    } catch (Exception exe) {
        throw new FactoryConnectionErrorException(exe.Message, exe); ;
    } finally {
        DataAccessManager.MyConection.Close();
    }

    return robots;
}
```

Extras:

Posee la facilidad de poder leer los distintos tipos de archivos que genera y consume el programa, tanto Txt, Binario y Xml. Luego de cargarlos, los muestra en un nuevo formulario.





En el warehouse, posee un buscador a modo de filtro que **no es case sensitive**, al empezar a tipear el origen o modelo a buscar, irá filtrando el resultado y mostrándolo en eldatagrid. (Hay un botón a su lado, pero es de adorno ya que se aprovechó el icono de dicho componente)

03:01:30
lunes, 12 de julio de 2021

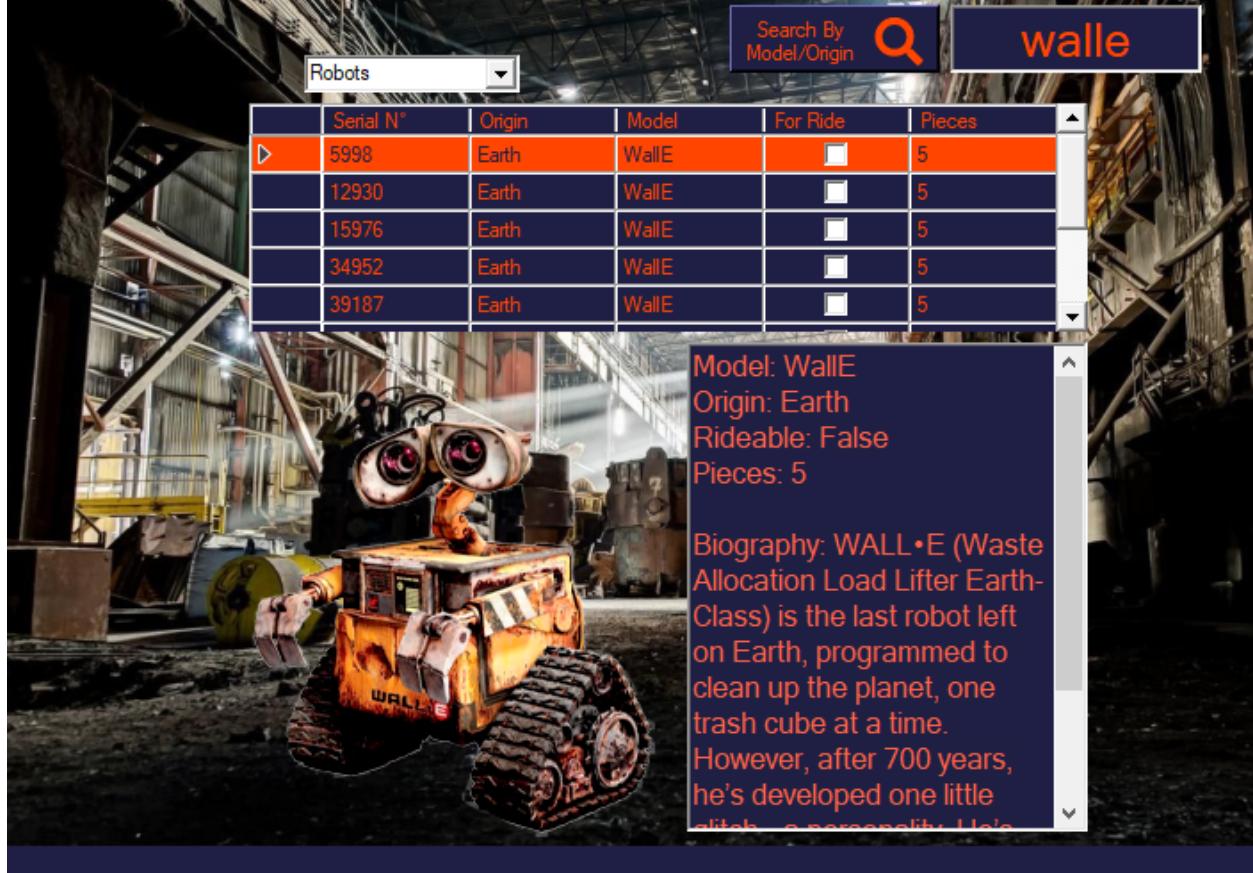
Open File

Warehouse

Search By Model/Origin  walle

Robots

	Serial N°	Origin	Model	For Ride	Pieces
▶	5998	Earth	WallE	<input type="checkbox"/>	5
	12930	Earth	WallE	<input type="checkbox"/>	5
	15976	Earth	WallE	<input type="checkbox"/>	5
	34952	Earth	WallE	<input type="checkbox"/>	5
	39187	Earth	WallE	<input type="checkbox"/>	5

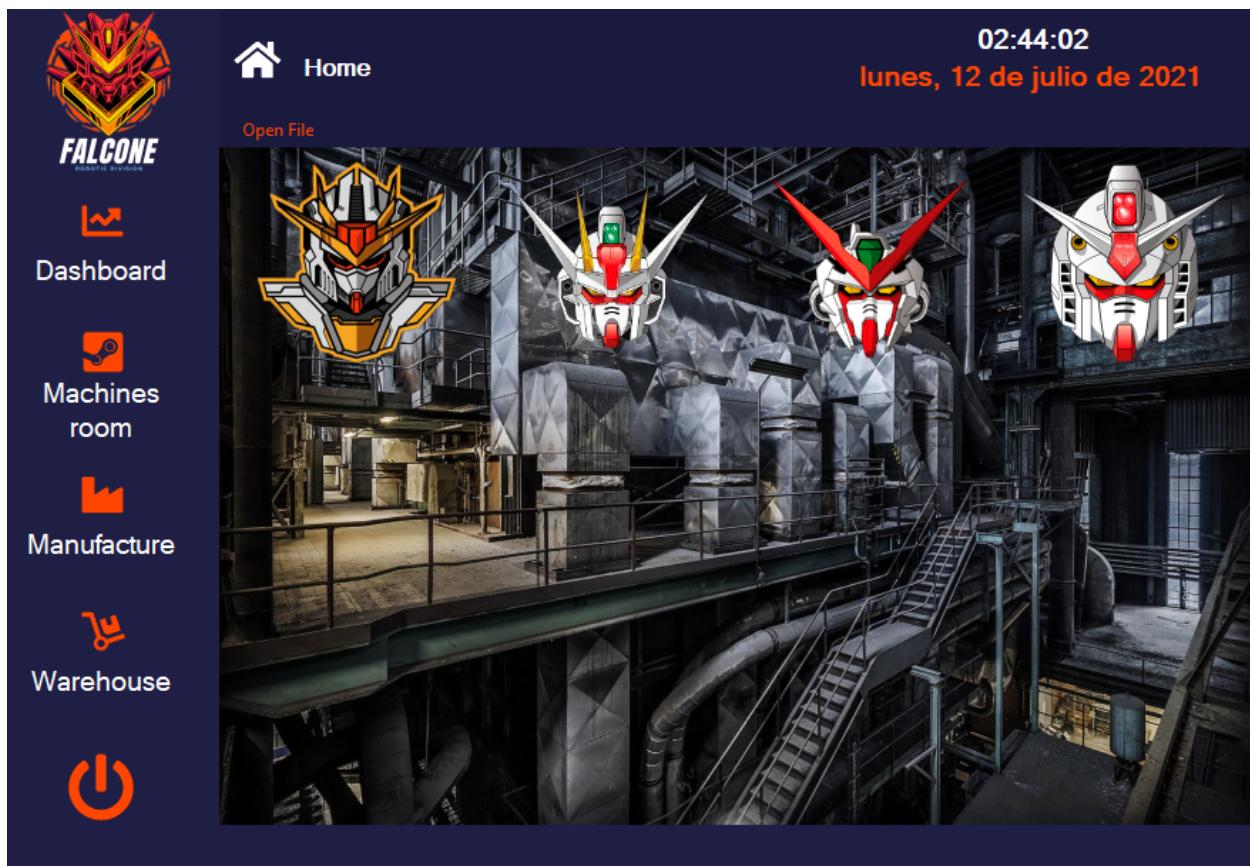


Model: WallE
Origin: Earth
Rideable: False
Pieces: 5

Biography: WALL•E (Waste Allocation Load Lifter Earth-Class) is the last robot left on Earth, programmed to clean up the planet, one trash cube at a time. However, after 700 years, he's developed one little *glitch*: a personality. He's

Secret Level:

Trata de encontrar un botón secreto escondido en la GUI principal del programa, Correlo y ve que pasa ;)



© Facu Falcone

Berazategui, Buenos Aires, Argentina, 2021