

Industriel Programmierung

Group 8





Problem



In industrial production & logistics environments, order handling is often performed manually with limited automation



This results in:

- Inefficient picking
- Operator dependency
- Higher error risk
- Variable quality and lead times



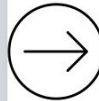
Therefore, there is a strong need for solutions that can automate picking from fixed pick-up positions



Solution

Solution

- Industrial program
- Programmed to pick-up and place components to complete order
- Using predefined pick-up and placements positions
- The system is able to complete orders
- Once completed, the admin must confirm



Program concept

- Minimal setup:
 - 4 fixed positions: 3 pick-up + 1 placement
 - Products per pick-up location, all different sizes
- Aligns with the KISS principle by keeping it minimal and simple.

Flowchart

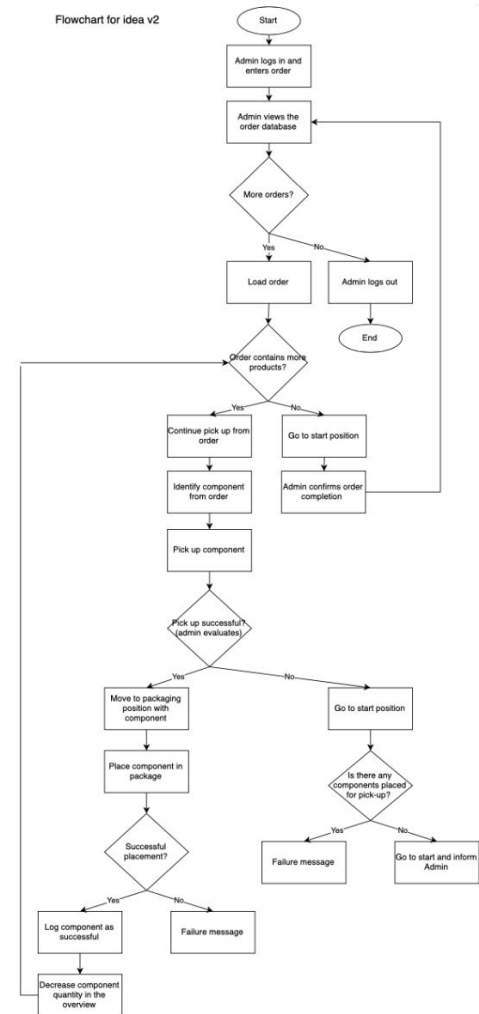
Flow chart:

- Definition
- Shapes

Order Flow (for admin)

- Logs in
- Creates order
- Process order
- Confirms order

Flowchart for idea v2



Flowchart

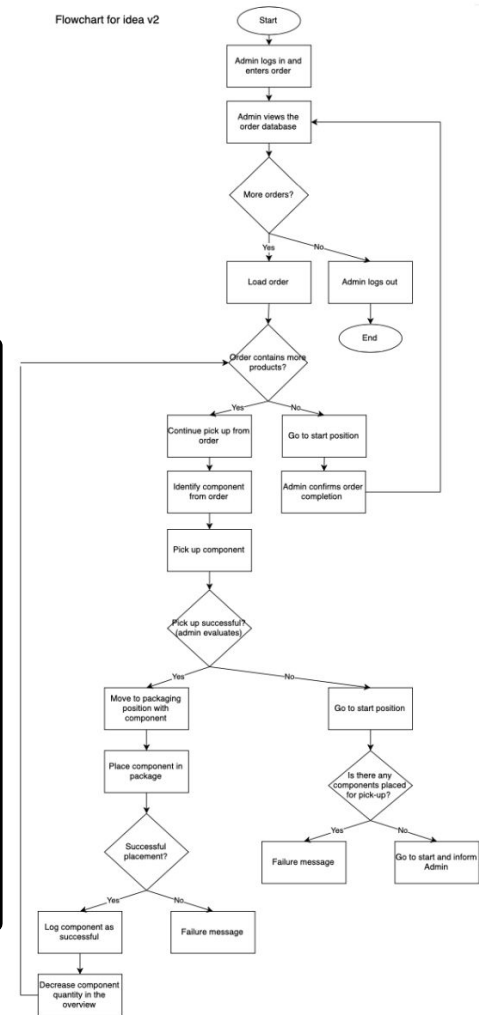
Flow chart:

- Definition
- Shapes

Order Flow (for admin)

- Logs in
- Creates order
- Process order
- Confirms order

Flowchart for idea v2



Flowchart

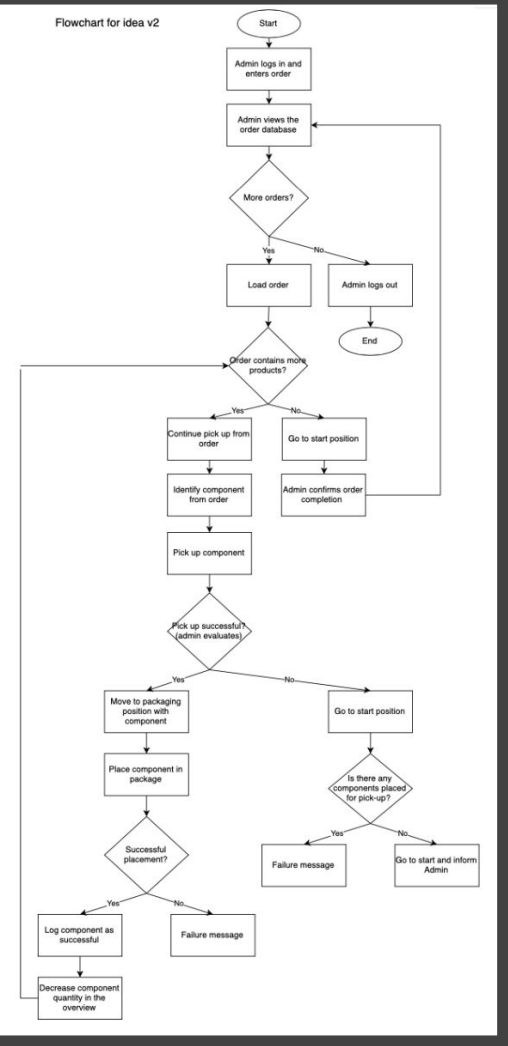
Flow chart:

- Definition
- Shapes

Order Flow (for admin)

- Logs in
- Creates order
- Process order
- Confirms order

Flowchart for idea v2





System

System Components

- C#: main programming language
 - GUI + Database
- URScript: robot logic for correct product handling

Design Principles

- Clear separation of responsibilities:
 - GUI → User input
 - C# logic → Validation and control
 - Robot → Physical execution
- Reduced risk of execution errors

Key Outcome

- Reliable and predictable order execution
- Automated robot behavior based on validated order data



File structure

```
SystemLogin · 1 project
├── SystemLogin
│   ├── Dependencies
│   ├── App.xaml
│   │   ├── App.xaml.cs
│   │   ├── app.manifest
│   │   ├── database.sqlite
│   │   ├── FlowController.cs
│   │   └── MainWindow.xaml
│   │       ├── MainWindow.xaml.cs
│   │       ├── Models.cs
│   │       ├── Program.cs
│   │       ├── RobotConnectionTest.cs
│   │       └── RobotOrderServer.cs
```

MainWindow.axaml

- GUI
- Displays what the user sees and interacts with

MainWindow.axaml.cs

- Logic behind the GUI
- Connects the UI with data and robot

database.sqlite

- Stores persistent data such as users, orders, and products
- Accessed via Entity Framework

FlowController.cs

- Flow Chart
- Process control and coordination

Models.cs

- Data models
- User, Order, Product
- Used by both the database layer and the GUI

RobotConnectionTest.cs

- URScript-based robot communication
- Contains the robot program logic, including:
 - Pick-and-place operations
 - Gripper logic

RobotOrderServer.cs

- Translates C# logic into URScript
- Sends robot movements based on the order

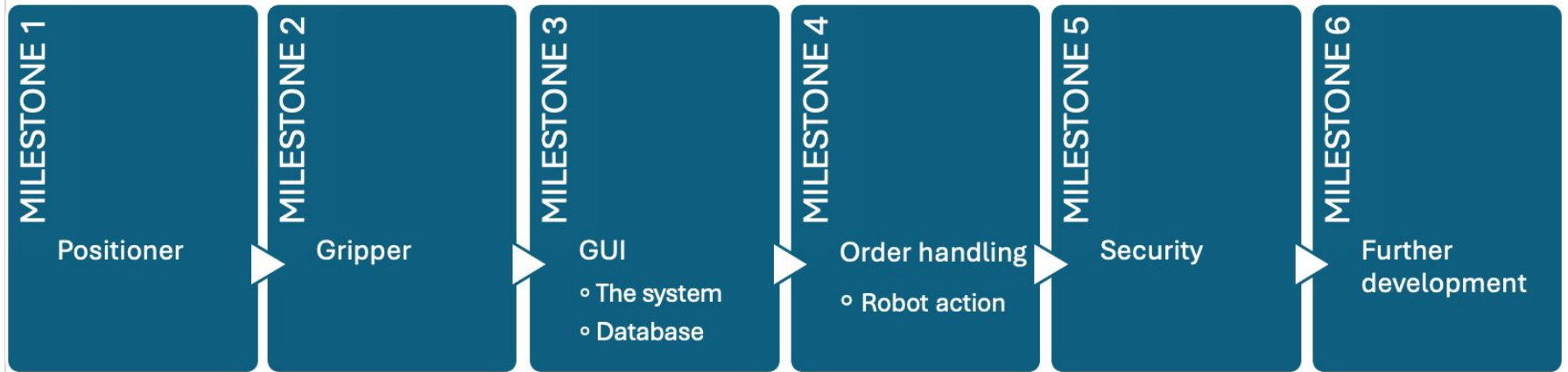


Video

<https://youtu.be/Z58rQTUVuj0>

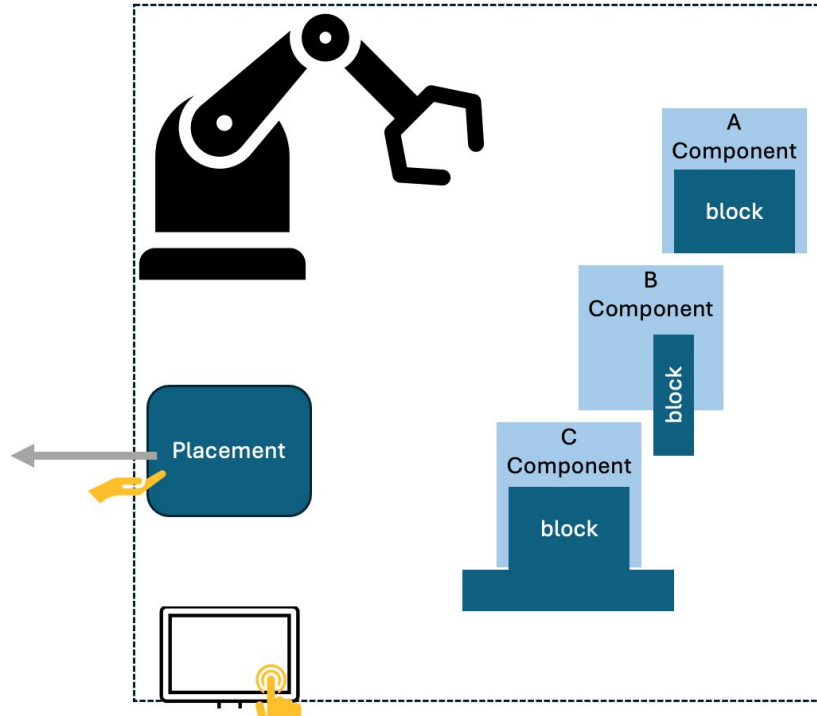


Design & Implementation



Milestone 1: Positions

Purpose: Defining the robot positions for the three pick-up positions as well as the placement



Process:

- Manually moved the robot arm to the desired locations
- Recorded the position values shown on the robot interface
- Evaluated and selected the positions and coordinates that best matched our solution

Milestone 1 - coding

```
"def prog():\n" +\n"  p1 = p[0.130,-0.345,0.548,2.01,-0.001,-0.007]\n" +\n"  p2 = p[0.482,-0.118,0.044,3.182,-0.003,-0.009]\n" +\n"  p6 = p[0.482,-0.118,-0.125,3.182,-0.003,-0.009]\n" +\n"  p3 = p[0.425,-0.225,0.044,3.146,-0.478,-0.001]\n" +\n"  p7 = p[0.425,-0.225,-0.125,3.146,-0.478,-0.001]\n" +\n"  p4 = p[0.292,-0.385,0.044,2.972,-1.166,-0.041]\n" +\n"  p8 = p[0.292,-0.385,-0.125,2.972,-1.166,-0.041]\n" +\n"  p5 = p[0.027,-0.482,0.044,2.508,-1.984,-0.015]\n" +\n"  p9 = p[0.027,-0.482,-0.05,2.508,-1.984,-0.015]\n"
```

```
Command="{Binding (MainWindow). Path= DataContext.ProcessOrderCommand,\n  RelativeSource={RelativeSource AncestorType=Window}}"/>
```

Position

p1

Start-position

p2, p3, p4

Pick-positioner (over komponent)

p6, p7, p8

Down-positioner (samme XY, lavere Z)

p5

Place-position (over)

p9

Place down-position

Knap der starter robotbevægelse

Brugeren trykker på knappen

ProcessOrderCommand kaldes

Command kalder robotkoden

Robotkoden bruger foruddefinerede positioner (p1–p9)

Milestone 2: Gripper

Purpose: To include gripper in the positions codes

Process:

- Implement in the position codes.
- Open → Close → Open
 - Open: Pick up
 - Close: Gripper around the component
 - Open: Component delivered

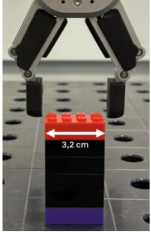
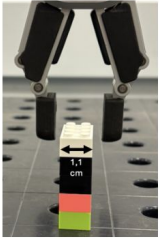
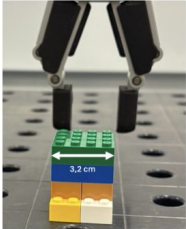
Component	Size for the gripper (components width)	Picture of the component
A component	3,2 cm	
B component	1,1 cm	
C component	3,2 cm	

Table: 3 - The components size



Milestone 2 - coding



The gripper is controlled in [RobotConnectionTest.cs](#)
Defined a reusable rg_grip() function

```
107         "    def rg_is_busy():\n" +  
108         "        return RPC.rg_get_busy(TOOL_INDEX)\n" +  
109         "    end\n" +
```

```
110     "    def rg_grip(width, force=20):\n" +  
111     "        RPC.rg_grip(TOOL_INDEX, width + 0.0, force + 0.0)\n" +  
112     "        sleep(0.01)\n" +  
113     "        while (rg_is_busy()):\n" +  
114     "            sleep(0.01)\n" +  
115     "        end\n" +  
116     "    end\n" +
```

Gripper status (busy-check)

"Ask" if there are any activity or the gripper is finished. Prevents that the robot move to early

Gripper-commands (open/ close)

width → How much the gripper can open/close

force → controls the gripper's gripping force - how hard the gripper clamps

while (rg_is_busy()) → keep running as long the condition is true - the robot repeats action

sleep → Pauses the program for a set amount of time - the robot waits before continuing



Milestone 3: GUI

Purpose: create a simple and functional setup that can store and manage the orders

GUI:

- Definition
- Purpose
 - Easier
 - Interaction

The system:

- Login tab
- Admin tab: previous orders
- Create order tab
- Database tab: placed orders, process and confirm
- Database

Database:

- GUI
 - C# logic
 - Stores data
 - URScript
 - Robot action



Milestone 3: GUI

Purpose: create a simple and functional setup that can store and manage the orders

GUI:

- Definition
- Purpose
 - Easier
 - Interaction

The system:

- Login tab
- Admin tab: previous orders
- Create order tab
- Database tab: placed orders, process and confirm
- Database





Database:

- GUI
 - C# logic
 - Stores data
 - URScript
 - Robot action

Milestone 3: GUI

Login tab

SystemLogin

 Login  Admin  Database  Create Order

Login

Username

Password


Login

Create user

Username

Password

Is admin ☐





Create user 

Clear log

Milestone 3: GUI

Admin tab

SystemLogin

 Login  Admin  Database  Create Order

Previous orders


Order ID	Date	Quantity
46	19.01.2026	4
13	16.01.2026	1
12	16.01.2026	1
11	16.01.2026	1
10	16.01.2026	1
9	16.01.2026	1
8	16.01.2026	1
7	15.01.2026	2
6	15.01.2026	3
5	15.01.2026	2
4	15.01.2026	1
3	15.01.2026	1
2	15.01.2026	2


Place new order


Milestone 3: GUI


Create order tab

SystemLogin

 Login




 Admin

 Database










 Create Order


Your information
Admin ID #
admin1

Order

Delete 	2	Component A
Delete 	2	Component B
Delete 	4	Component C

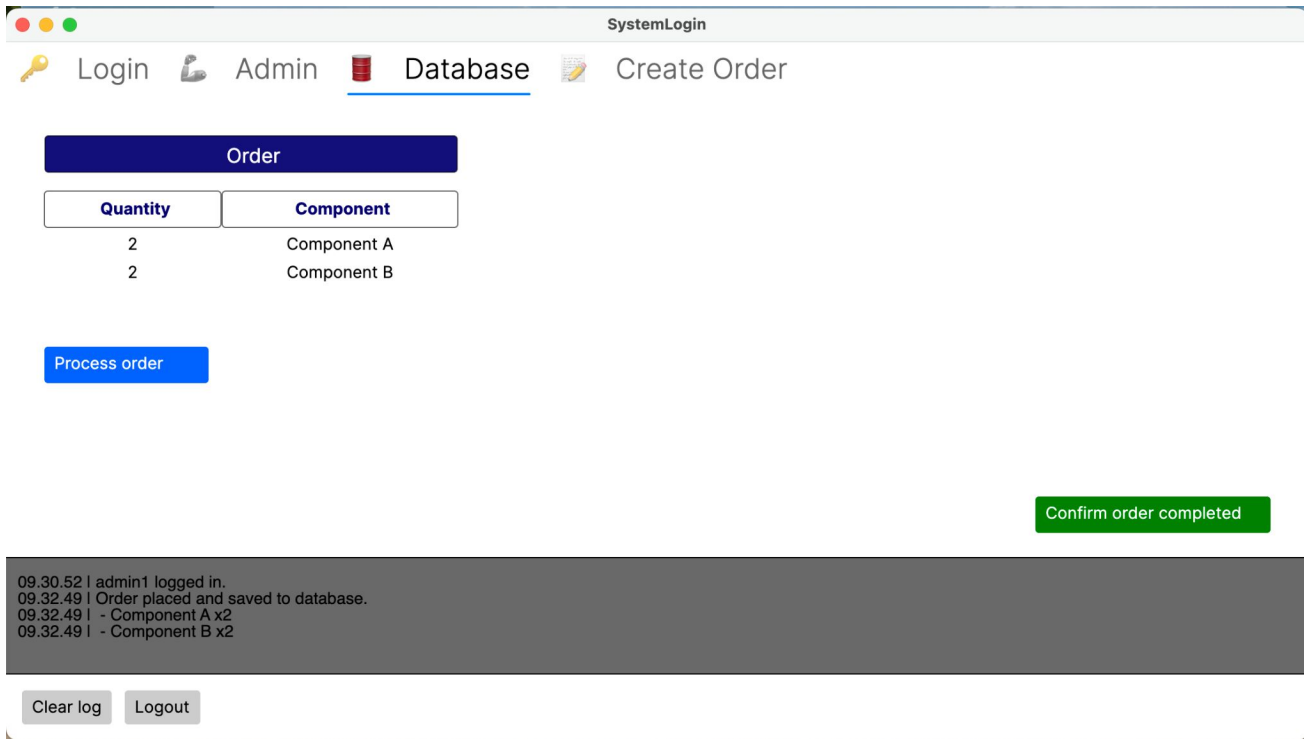
Products

		0	Component A	Add to order 
		0	Component B	Add to order 
		0	Component C	Add to order 

Place order 

Milestone 3: GUI

Database tab



The screenshot shows a web application window titled "SystemLogin". The navigation bar includes links for "Login", "Admin", "Database" (which is the active tab), and "Create Order". The main content area features a dark blue header labeled "Order" above a table. The table has two columns: "Quantity" and "Component". It lists two items: 2 units of "Component A" and 2 units of "Component B". Below the table is a blue "Process order" button. A green notification box in the bottom right corner states "Confirm order completed". The footer contains a log area with the following entries: "09.30.52 | admin1 logged in.", "09.32.49 | Order placed and saved to database.", "09.32.49 | - Component A x2", and "09.32.49 | - Component B x2". At the bottom left are "Clear log" and "Logout" buttons.

SystemLogin

Login Admin Database Create Order

Order

Quantity	Component
2	Component A
2	Component B

Process order

Confirm order completed

09.30.52 | admin1 logged in.
09.32.49 | Order placed and saved to database.
09.32.49 | - Component A x2
09.32.49 | - Component B x2

Clear log Logout



Milestone 3: GUI

Purpose: create a simple and functional setup that can store and manage the orders

GUI:

- Definition
- Purpose
 - Easier
 - Interaction

The system:

- Login tab
- Admin tab: previous orders
- Create order tab
- Database tab: placed orders, process and confirm
- Database

Database:

- GUI
 - C# logic
 - Stores data
 - URScript
 - Robot action



Milestone 3 - coding, login example

AXAML:

1. The user enters a username and password.
2. When the Login button is clicked, the `LoginButton_OnClick` method is invoked.
3. The user credentials are validated against the database.

```
59         <TabItem Header="🔑 Login">
60             <StackPanel Margin="20" Spacing="20">
61                 <TextBlock Text="Login" FontWeight="Bold" FontSize="16"/>
62                 <TextBox Name="LoginUsername" Watermark="Username"/>
63                 <TextBox Name="LoginPassword" Watermark="Password" PasswordChar="●"/>
64                 <Button Content="Login" Width="160" Click="LoginButton_OnClick"/>

```

AXAML.CS:

1. This is an async click-event handler that runs when the Login button is pressed;
2. `object? sender` is the clicked UI element and that can be = 0

```
179     private async void LoginButton_OnClick(object? sender, RoutedEventArgs e)
180     {

```



Milestone 4: Order handling

Purpose: establish an order flow from the GUI to the robot

Industry 5.0 element

- Human interaction included through GUI:
 - Login
 - Create new order
 - Start/confirm order
- Physically restock the components



Milestone 4 - coding



RobotConnectionTest.cs

```
72     private static string BuildOrderProgram(int qtyA, int qtyB, int qtyC)
73     {
```

```
76         var run = "";
77         for (int i = 0; i < qtyA; i++) run += " do_A()\n";
78         for (int i = 0; i < qtyB; i++) run += " do_B()\n";
79         for (int i = 0; i < qtyC; i++) run += " do_C()\n";
```

- The admin places an order.
- `BuildOrderProgram(qtyA, qtyB, qtyC)` receives the order.
- For each component, a for-loop repeats the corresponding robot action: `do_A`, `do_B`, `do_C`.
- Each `do_X()` function executes a complete pick-and-place sequence: move to pick → grip → move to place → release.
- The generated URScript is sent to the robot, which executes the actions in the exact order and quantity specified.

```
119     " def do_A():\n" +
120     "     movej(p1, a=a, v=v)\n" +
121     "     movej(p2, a=a, v=v)\n" +
122     "     movel(p6, a=a, v=v)\n" +
123     "     rg_grip(50)\n" +
124     "     rg_grip(32)\n" +
125     "     movel(p2, a=a, v=v)\n" +
126     "     movej(p5, a=a, v=v)\n" +
127     "     movel(p9, a=a, v=v)\n" +
128     "     rg_grip(50)\n" +
129     "     movel(p5, a=a, v=v)\n" +
130     "     movej(p1, a=a, v=v)\n" +
131     " end\n" +
```



Milestone 5: Security



Purpose: To ensure that security is integrated into our project

Security:

- Definition
 - Protecting
 - Access
- Why is it important?
 - Equipment
 - Downtime

Implemented principles:

- KISS
 - Overall
- Hash function
 - Model.cs
- Principle of least privilege
 - Model.cs

Effect on our project:

- Maintain
- Secure against unauthorized access



Milestone 5: Security



Purpose: To ensure that security is integrated into our project

Security:

- Definition
 - Protecting
 - Access
- Why is it important?
 - Equipment
 - Downtime

Implemented principles:

- KISS
 - Overall
- Hash function
 - Model.cs
- Principle of least privilege
 - Model.cs

Effect on our project:

- Maintain
- Secure against unauthorized access



Milestone 5: Security



Purpose: To ensure that security is integrated into our project

Security:

- Definition
 - Protecting
 - Access
- Why is it important?
 - Equipment
 - Downtime

Implemented principles:

- KISS
 - Overall
- Hash function
 - Model.cs
- Principle of least privilege
 - Model.cs

Effect on our project:

- Maintain
- Secure against unauthorized access



Milestone 5: Security

Hashing

Password creation:

```
93  ✓    public (byte[] Salt, byte[] Hash) Hash(string password)
94      {
95          var salt = RandomNumberGenerator.GetBytes(saltLength);
96          return (salt, Hash(salt, password));
97      }
```

- Create account
- Password + salt = hash
- Stored

Password verification:

```
74      public bool PasswordCorrect(string password, byte[] salt, byte[] saltedPasswordHash)
75      {
76          return CryptographicOperations.FixedTimeEquals(Hash(salt, password), saltedPasswordHash);
77      }
```

- Login -> password + salt = hash
- Compare



Milestone 5: Security

Principle of least privilege

User role:

125 `public bool isAdmin { get; set; }`

- Role

Checks permissions:

57 `return db.Accounts.Where(a => a.Username == username).Select(a => a.isAdmin).FirstAsync();`

- Privileges
- queries isAdmin



Future improvements

Sensor-based solution

- Determine whether there are components left in each storage position

→ System more reliable and optimize the process

Implementation of a clear abort action

- Would allow the system to stop safely and reset in a controlled way

→ Solution more robust in operation

Using BendArray

- Creating smoother transitions between movements

→ Reduce:

- Motion changes
- Optimize cycle time
- Improve overall efficiency



Conclusion

Purpose: Develop an automated order-handling solution for an industrial set up.



Solution: Integrates robot, GUI, database and order-handling logic



Orders can be created and processed automatically
The robot performs pick-and-place operations



Human involvement is still required → Supports Industry 5.0 perspective



Some milestone were adjusted due to technical limitations - Improvements led to better robot movements



The project provided valuable insights into industrial programming.