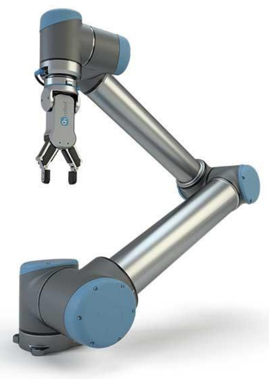
Group 1
Lars Bach Sørensen (S235648) ,
Lasse Manicus (S235655),
Marius Millington (S235659)

INDUSTRIAL PROGRAMMING



Group 1
Lars Bach Sørensen (S235648) ,
Lasse Manicus (S235655),
Marius Millington (S235659)

INDUSTRIAL PROGRAMMING

TOY FACTORY

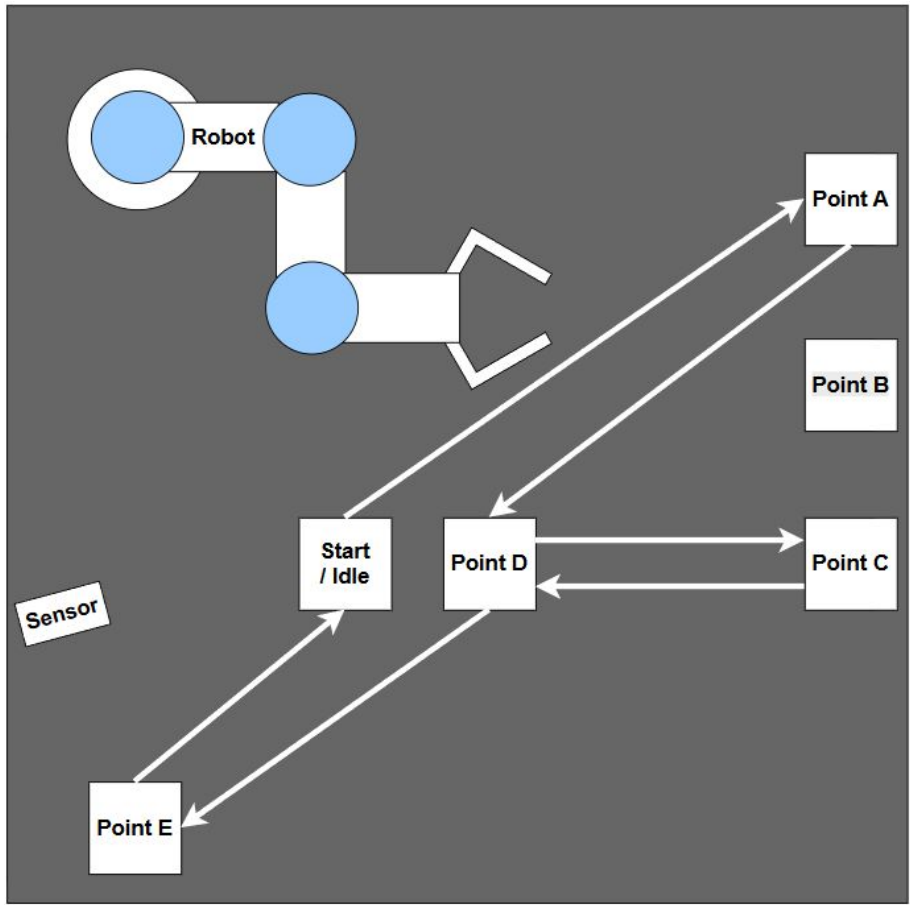
Introduction / Problem (Scope)

Project context

- Automated assembly prototype for a toy box factory
- Developed as part of the *Industrial Programming* course
- Focus on integrating software, database, and an industrial UR robot

Project objective

- Convert digital production orders into deterministic robot motion
- Demonstrate end-to-end flow:
  GUI → Database → Robot → Database
- Ensure predictable, safe, and repeatable robot behavior



Demonstration video

Robot demonstration - Group 1

System Architecture

GUI / Operator Station

- Create & monitor queue & orders
- Start / control production

Application Layer (Control Logic)

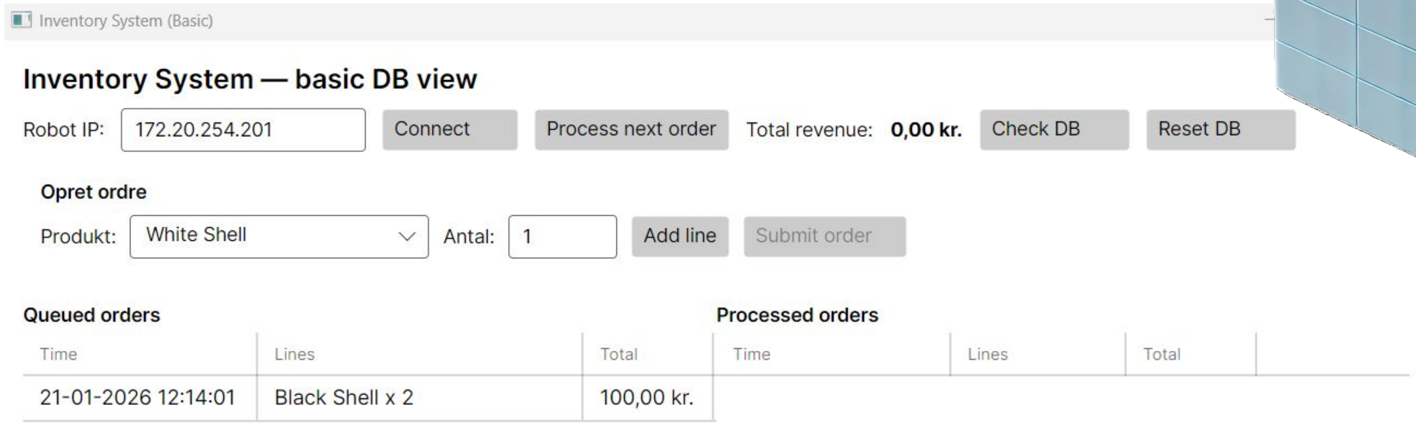
- Reads next queued order
- Maps order → predefined robot sequence
- Executes robot + update DB

Data Layer (EF Core + SQLite)

- Orders (Queued / Processed)
- Persistent system state

Robot Integration Layer

- URScript via TCP/IP
- Centralized motion sequences & positions



Inventory System (Basic)

Inventory System — basic DB view

Robot IP: 172.20.254.201 | Connect | Process next order | Total revenue: **0,00 kr.** | Check DB | Reset DB

Opret ordre

Produkt: White Shell | Antal: 1 | Add line | Submit order

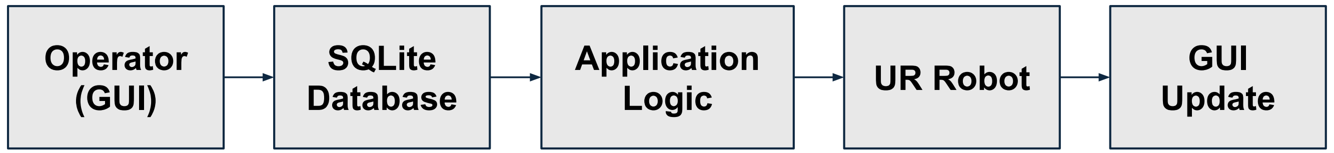
Queued orders

Time	Lines	Total
21-01-2026 12:14:01	Black Shell x 2	100,00 kr.

Processed orders

Time	Lines	Total

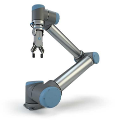
Production flow:



Status: Order submitted ✓

Design Principles

- Single source of truth: Database
- Clear separation of layers
- Traceability & restart safe operation
- Robot logic decoupled from UI & data

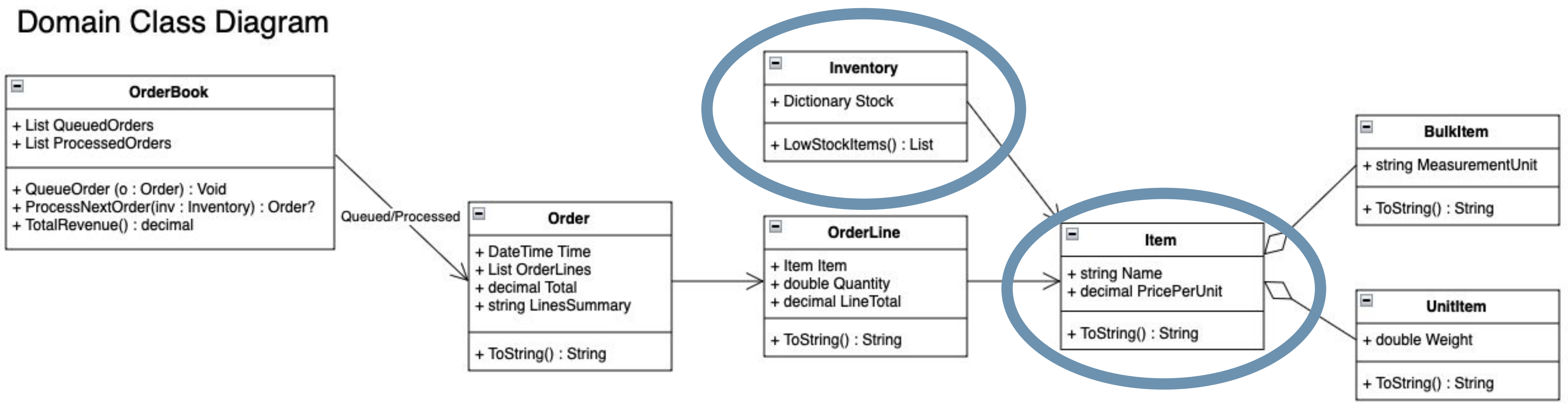


Domain / Inventory

Domain Model – Core Business Logic:
What is produced and in which order, rather than robot assembly. Independent of gui and robot.

Inventory & item

- Represents physical components used in production
- Inventory = collection of items + quantities
- Common item abstraction with optional specialization
- Tracks stock an updates when orders are processed.



Domain / Inventory

OrderBook & Order States

- Central production flow controller

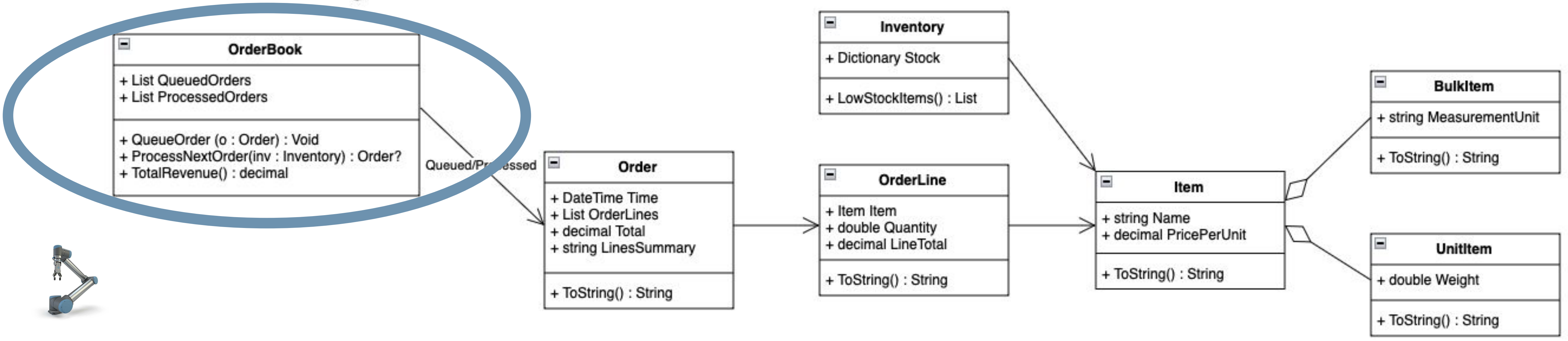
Two states

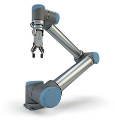
- Queued Orders (pending)
- Processed Orders (completed)

Characteristics

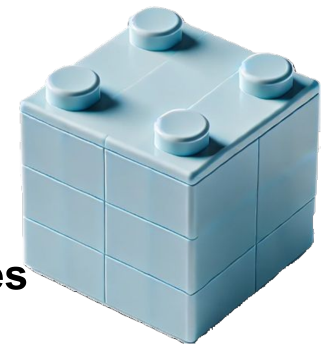
- **Operator choice:** the GUI lets the operator *compose* an order (White/Black shell lines) and submit it.
- **Queue rule:** once submitted, orders are **queued** and executed **FIFO**.
- **Traceability:** queue/processed state is stored in the database (single source of truth).

Domain Class Diagram





Domain / Inventory



Design Rationale

- Domain logic isolated from UI and robot execution
- Operator control improves transparency, safety, and testing
- Deterministic execution with option for future automation

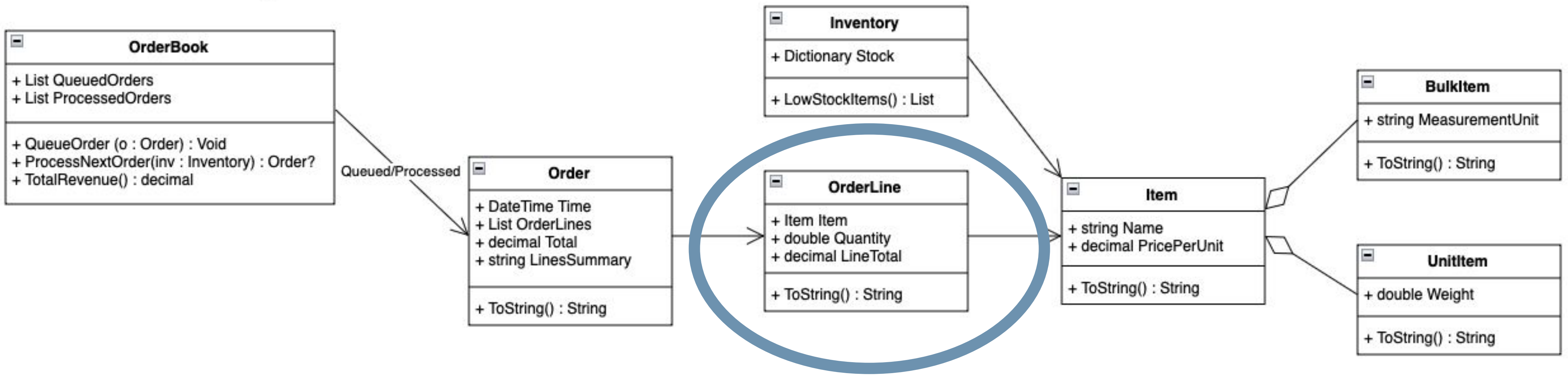
Result:
A simplified but realistic production control model combining **human supervision + data-driven automation**

Order Model

- **Order** = timestamp + list of **OrderLines**
- **OrderLine** = item + quantity
- Supports multi-item orders and future product expansion

Technical Example: Process Next Order -> Boolean (order queued condition)-> if else (stop or continue) -> Loop (based on quantity). Strings for items (reference text)

Domain Class Diagram



Database Design

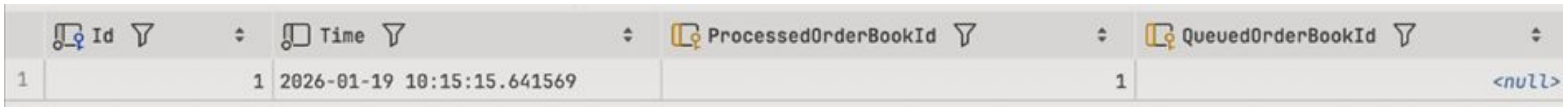
Database Technology and Access

- Ensures consistency between GUI, Application logic
- and physical robot execution.
- SQLite
- Operator controlled system (Human vs full automation)
- Entity Framework Core to persist production state
- Seeding & Reset

Data Flow & Consistency

- All orders, inventory data, and production state are stored in the database

- Application logic reads this data and translates it into robot motion sequences.



	Id	Time	ProcessedOrderBookId	QueuedOrderBookId
1	1	2026-01-19 10:15:15.641569	1	<null>

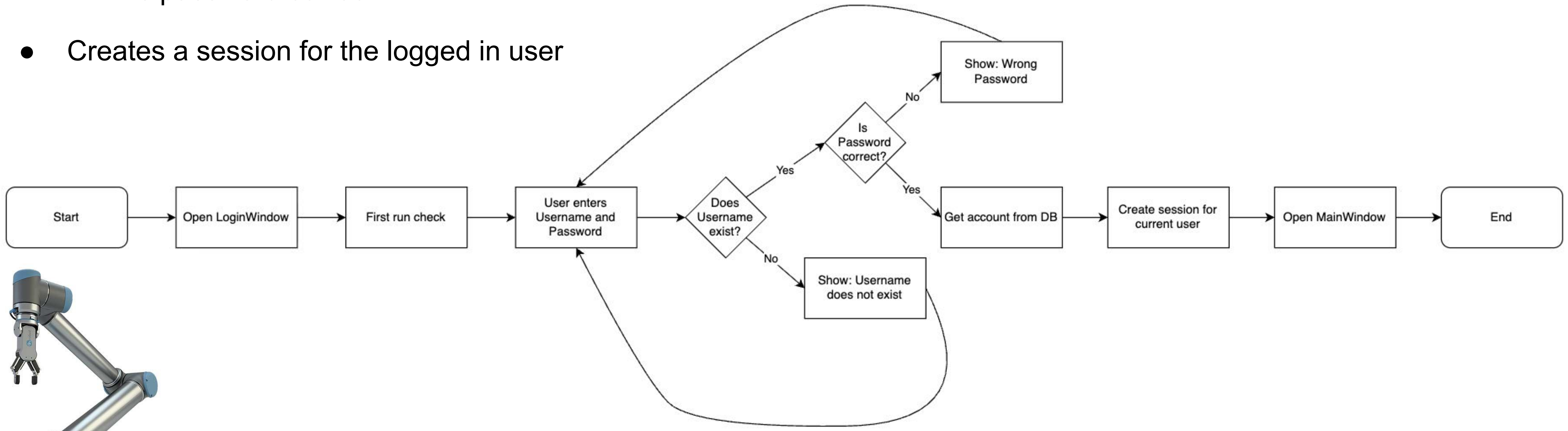
Security

Login feature

- Database controlled
- Has multiple verification levels
  - Does Username exist
  - Is password correct
- Creates a session for the logged in user

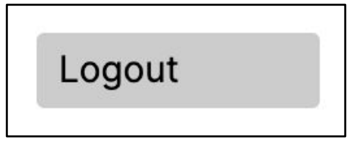
Password Salting

- The system uses salting in database
- Protects against precomputed attacks like Rainbow table

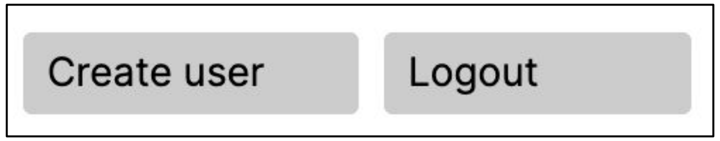


Security

User



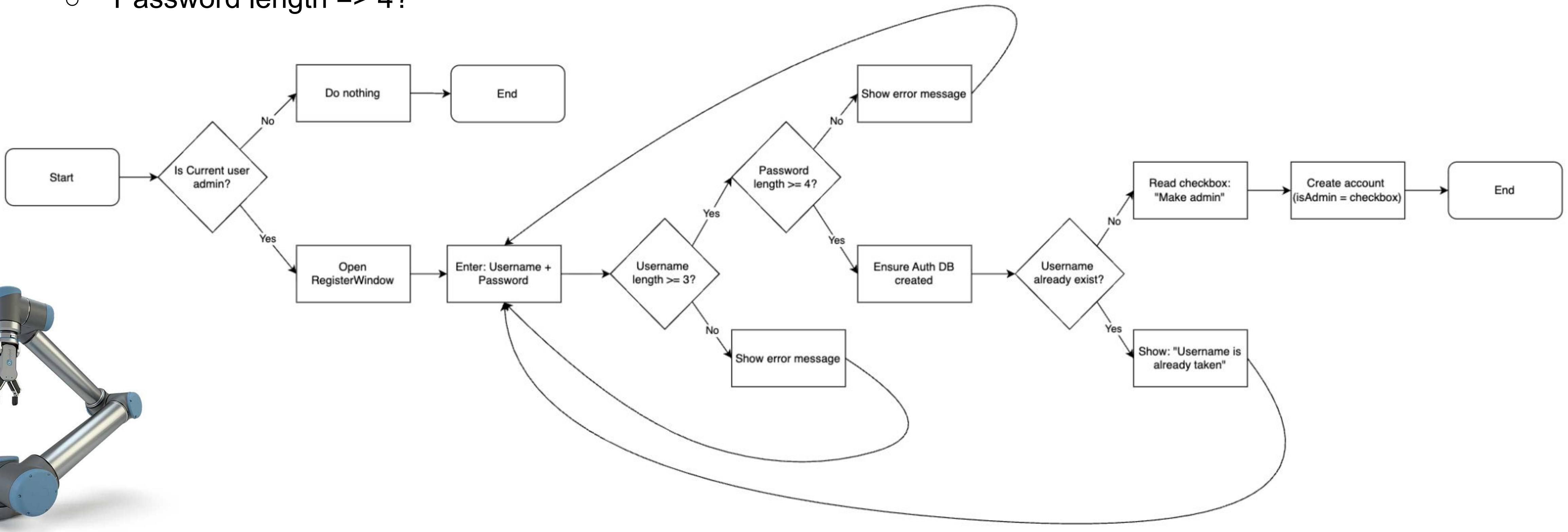
Admin User

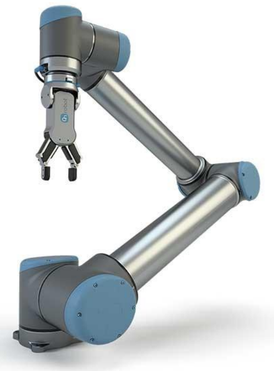


Create User (Admin)

- Create new user is limited to admins
- Account requirements
    - Username length => 3?
    - Password length => 4?

- Option to make admin
- Creates account in database



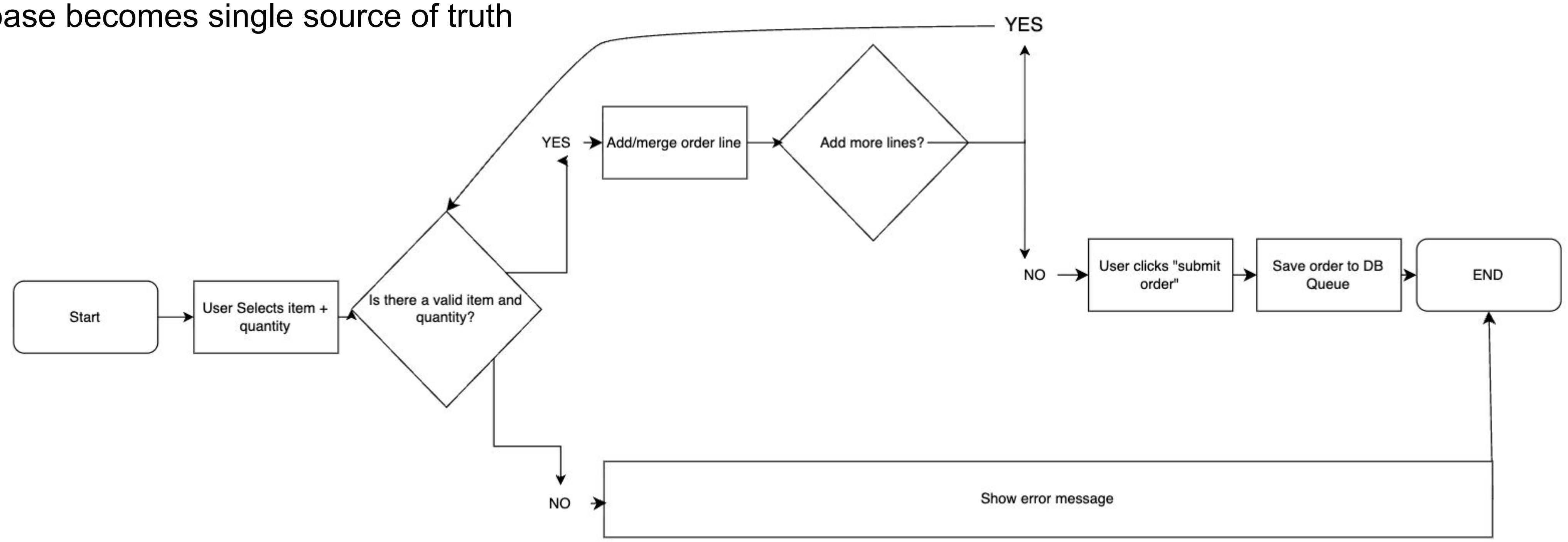


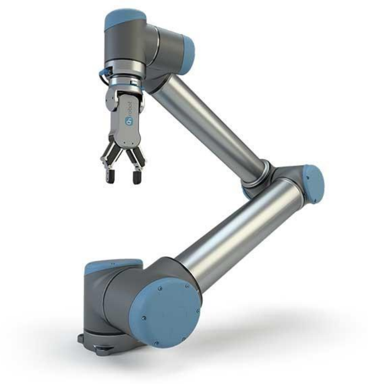
GUI and ViewModel (MVVM)

Interactive GUI able to both create orders and process orders.

Create order

- User able to select both item and quantity
- The order is first validated by the system
- Database stores as a queued order
    - Database becomes single source of truth



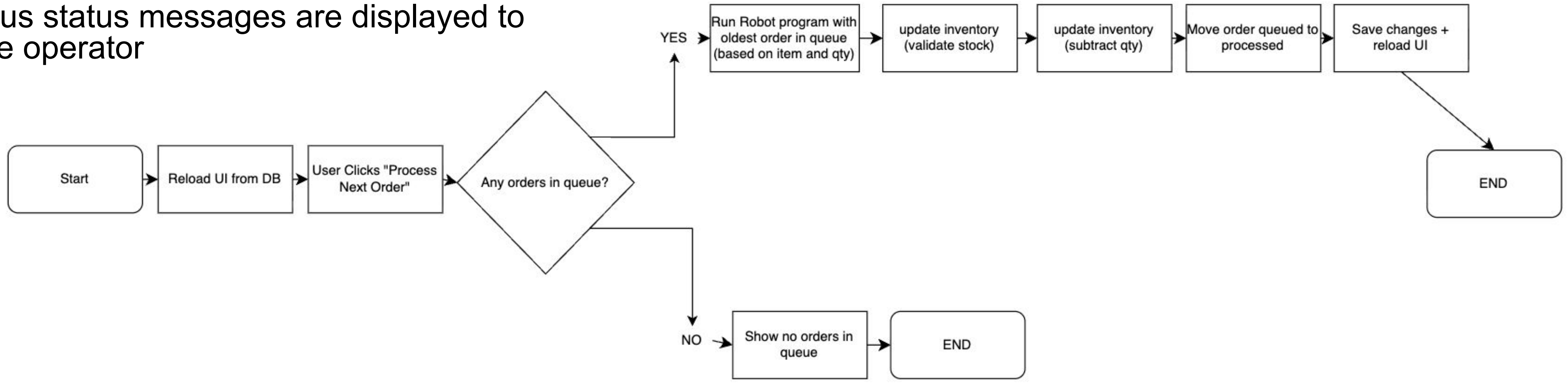


GUI and ViewModel (MVVM)

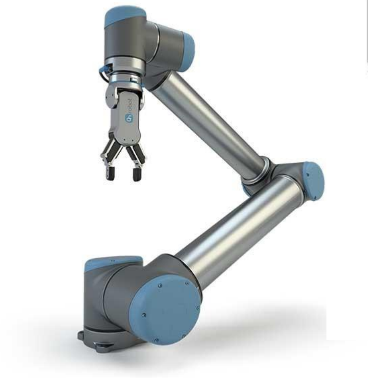
Interactive GUI able to both create orders and process orders.

Process order

- *Process Next Order* retrieves oldest queued order in database
- The order is translated to the robot
- When robot is done - Database is updated
  - Inventory updated
  - Queued → Processed
- Continuous status messages are displayed to inform the operator

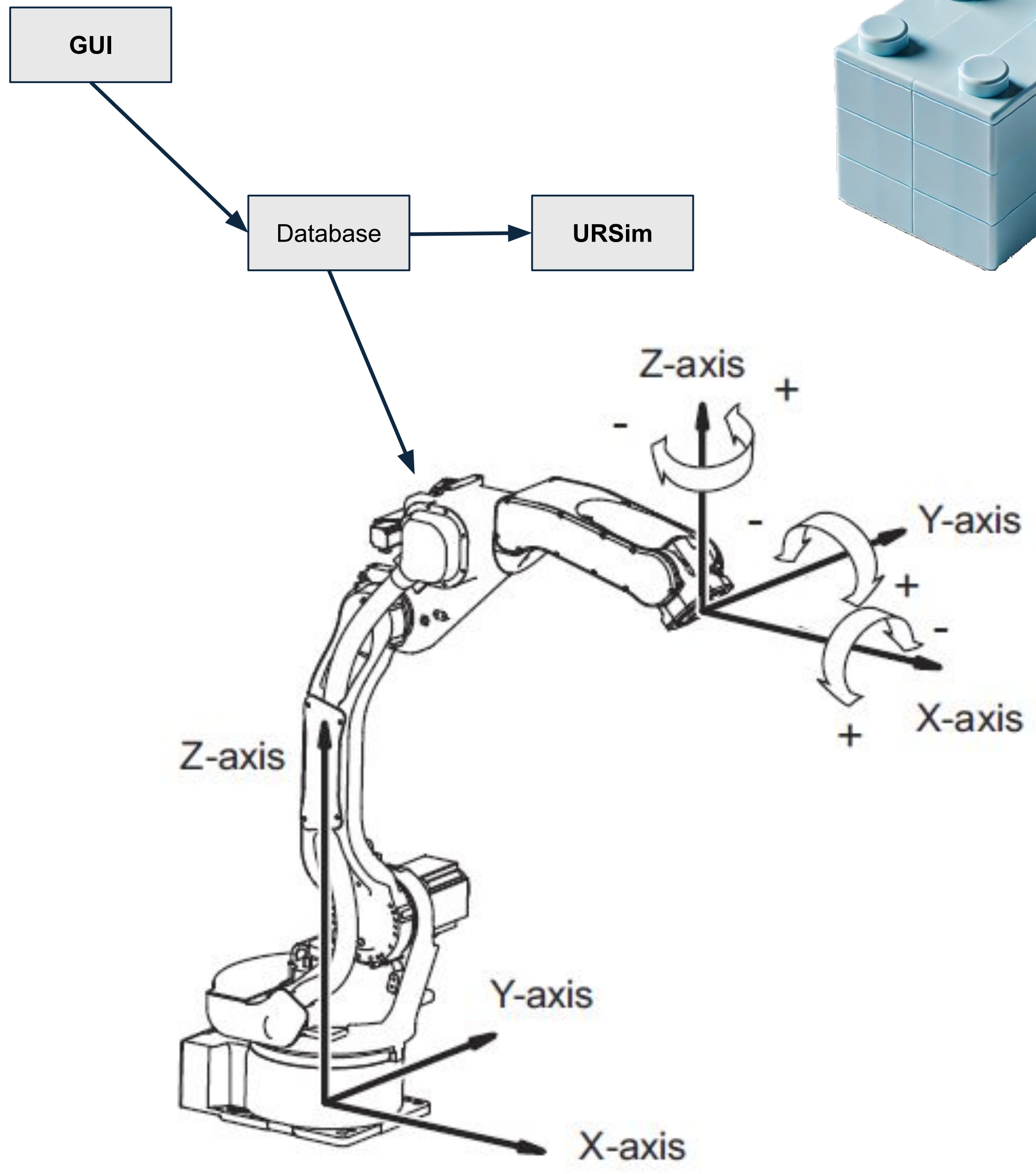


Start
Reload UI from DB
User Clicks "Process Next Order"
Any orders in queue?
YES
Run Robot program with oldest order in queue (based on item and qty)
update inventory (validate stock)
update inventory (subtract qty)
Move order queued to processed
Save changes + reload UI
END
NO
Show no orders in queue
END



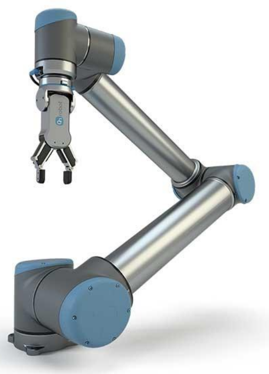
Robot Integration

- Database driven
- Orders created in GUI
- Stored in Database
- Communicating via
  URScript → TCP/IP → Ethernet → Robot
- Connects to URSim & real robot



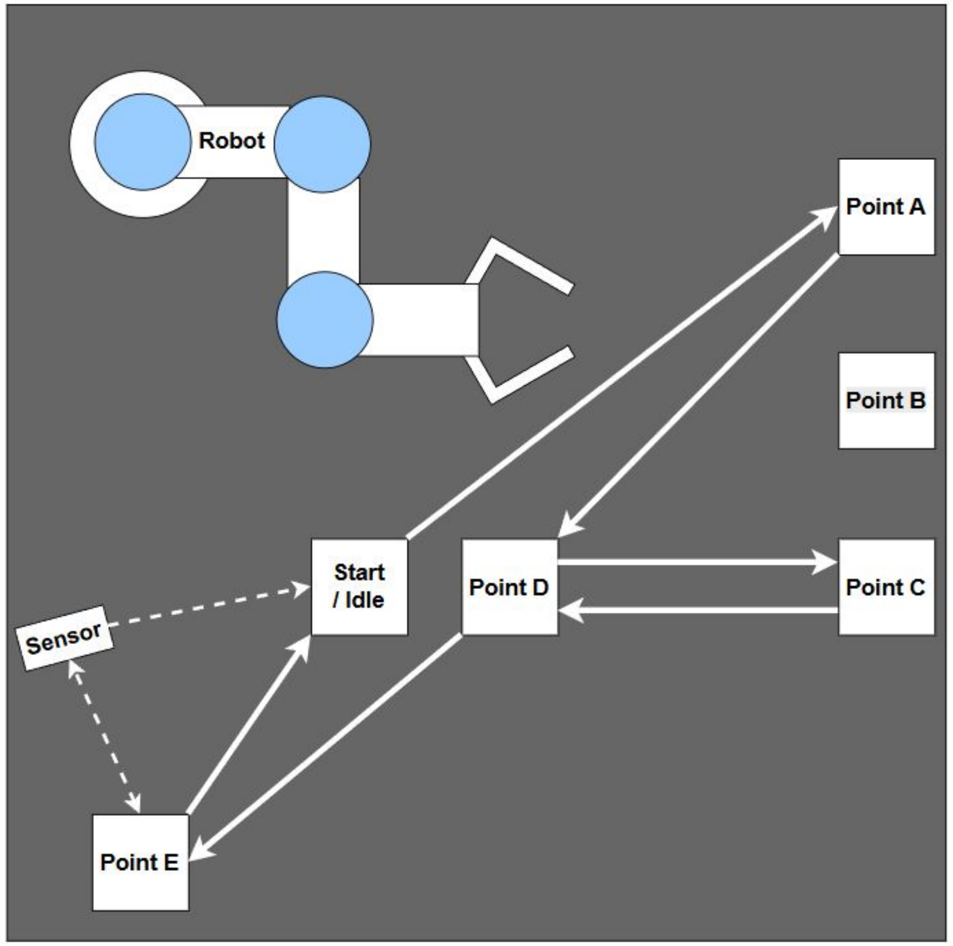
Positioning

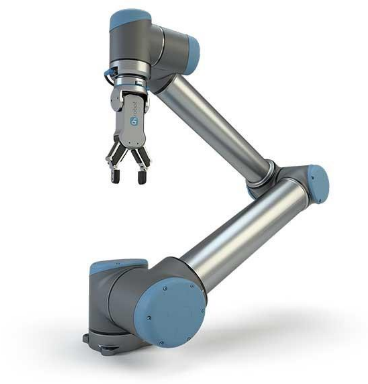
- Start position (<u>T</u>ool <u>C</u>enter <u>P</u>oint)
  - X = 0.125
  - Y = - 0.300
  - Z = 0.100
- Start rotation (<u>T</u>ool <u>C</u>enter <u>P</u>oint)
  - RX = 3.14 (180° nedad)
  - RY = 0.00
  - RZ =0.00



Robot Integration

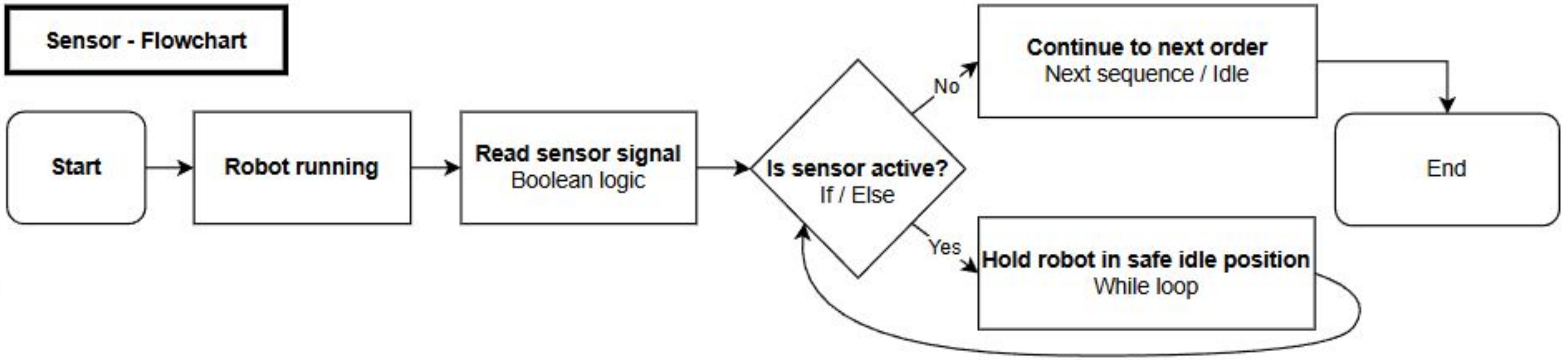
- Robot motion structure
  - Determined motion from A→E
  - Using Move L
- Pick & place - Height
  - Gripper - RG2
    - open_mm = 60
    - open_pick_mm= 85
    - close_mm = 31
    - f_open = 30
    - f_close = 30
- Predefined predictable sequences

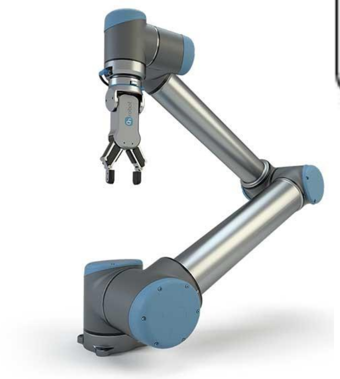




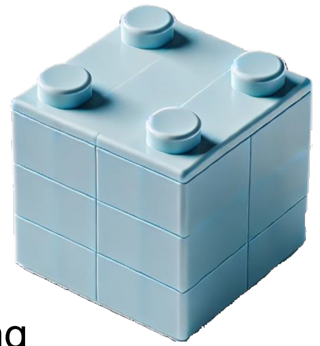
Sensor (Safety)

- Why safety is needed?
  - Due to human interaction
  - Error minimizing in production loop
- Sensor
  - Photoelectric
  - Detects products
  - Signals robot to wait or proceed
- Uses robot logic





Discussion



What worked well

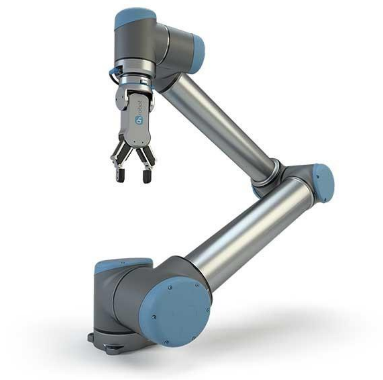
- Easy operator-controlling
- Stable robot communication
- Predictable robot behavior due to fixed positions
- Clear separation between GUI, database, and robot logic
- Easy testing using URSim

Challenges

- Robot calibration required significant testing
- Fixed positions reduce flexibility
- System relies on operator-controlling

Trade-offs

- Manual operator control chosen over full automation
- Fixed positions instead of vision system
- Safety prioritized over production



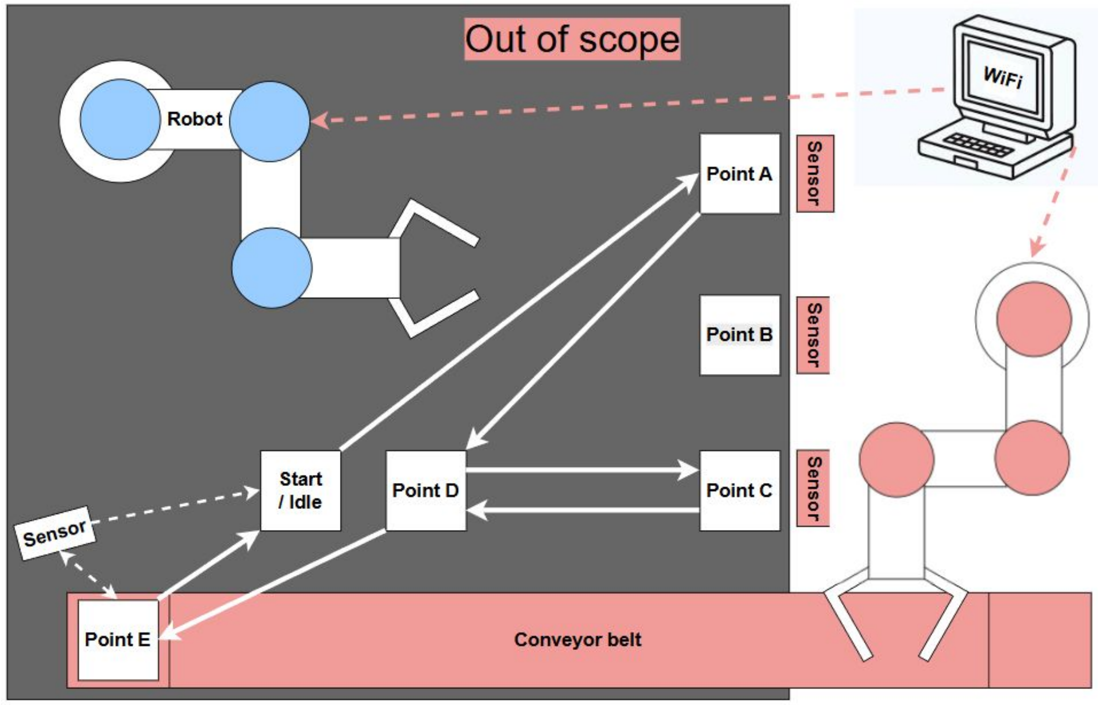
Solution

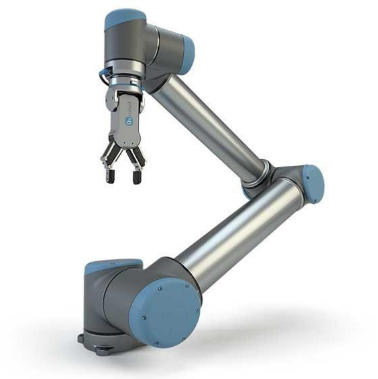
- Functional automated assembly prototype with human interaction
- End-to-end digital flow:
  - GUI → Database → Robot → Database
- Modular and maintainable architecture

Future implementation

Minimize human interaction

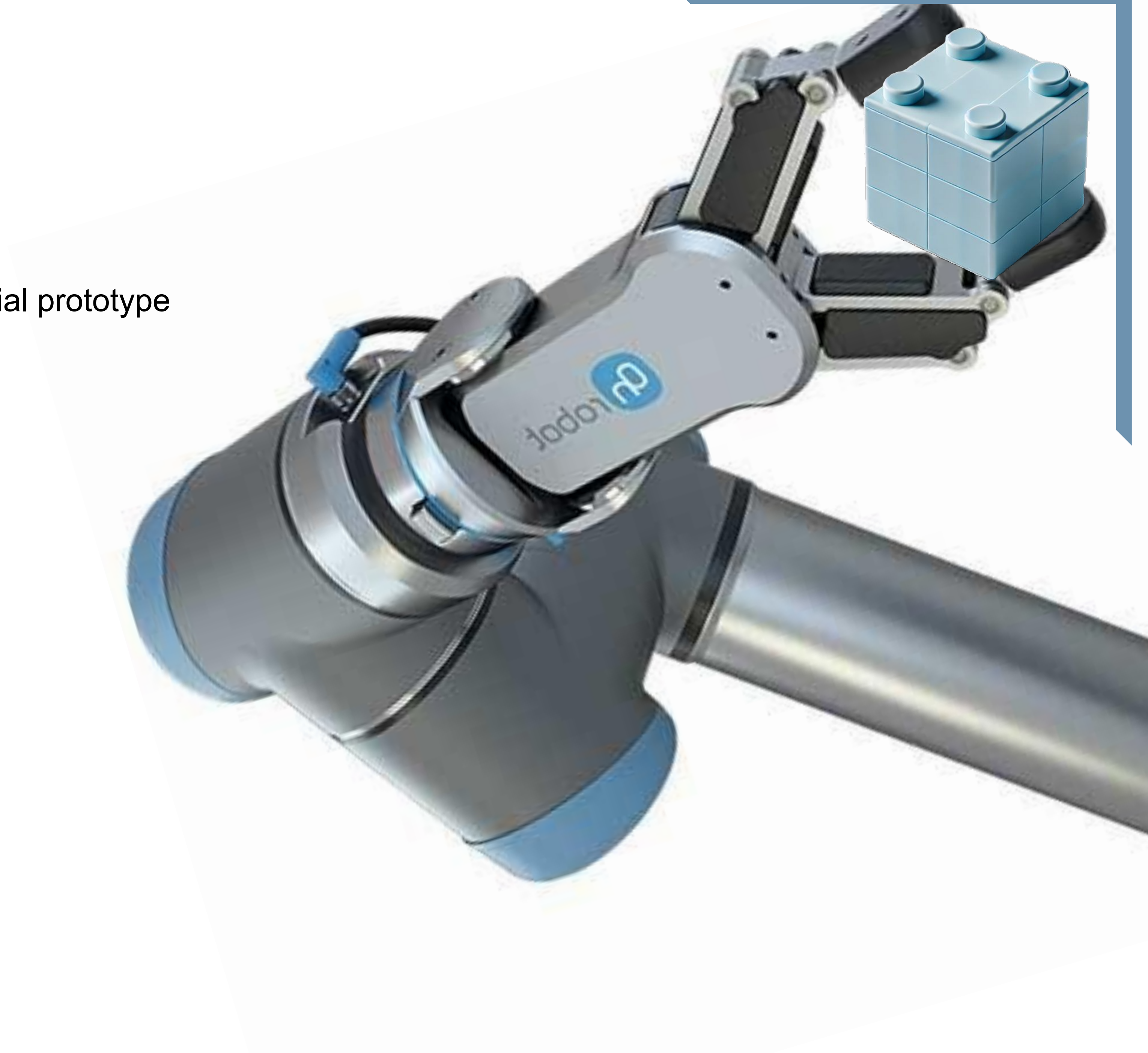
- Robot automatically replenish A, B & C
- Conveyor belt moves finished product
- Machine vision senses location/size



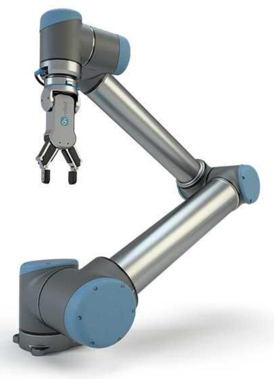


Conclusion

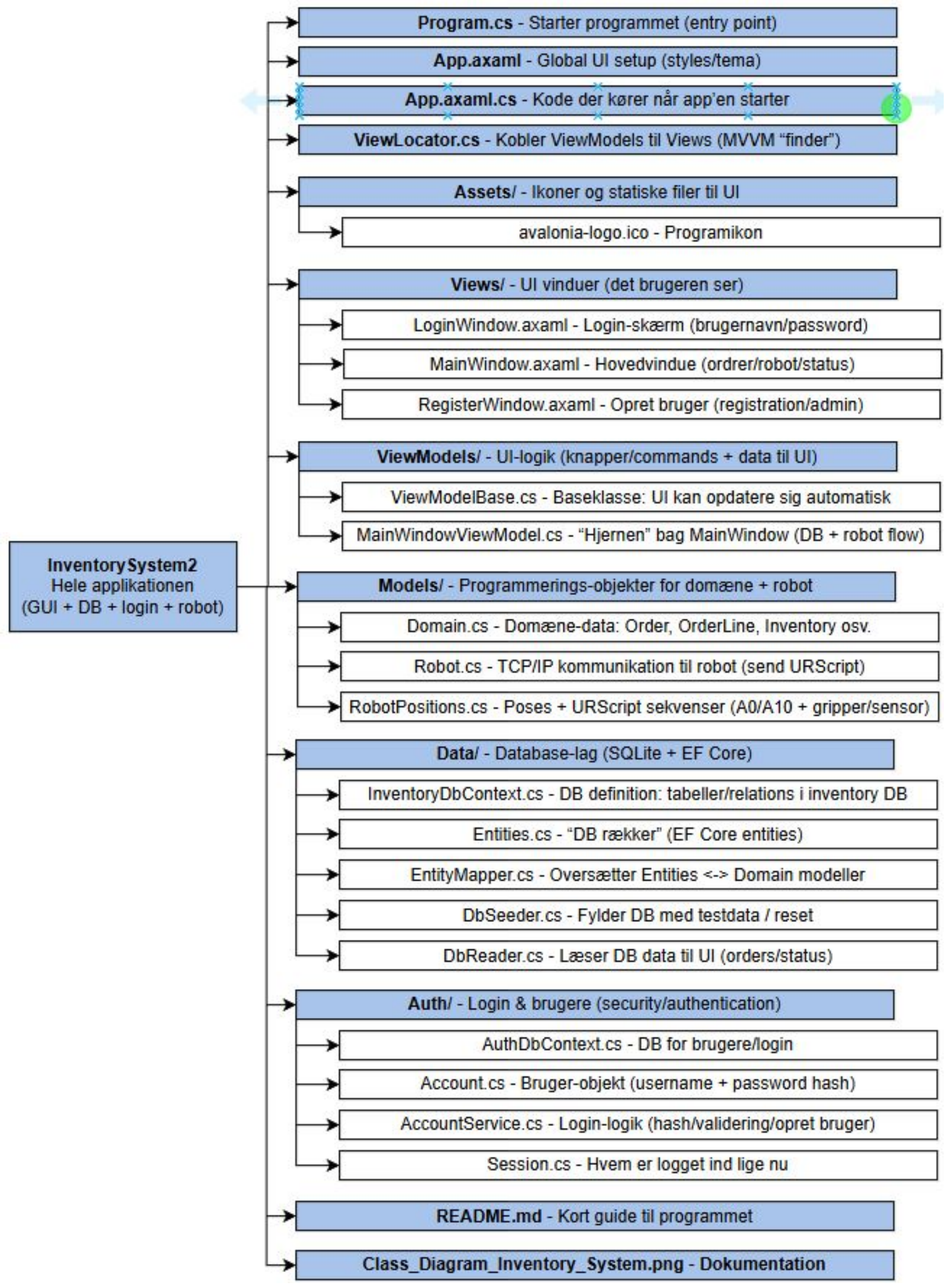
- Project objectives were met
- System behaves like a realistic industrial prototype
- Demonstrates principles from:
  - Industrial programming
  - Industry 4.0 & 5.0
  - Automation
  - Security & safety
  - Modular software design



Thank you!



Program overview


InventorySystem2 Hele applikationen (GUI + DB + login + robot)
Program.cs - Starter programmet (entry point)
App.axaml - Global UI setup (styles/tema)
App.axaml.cs - Kode der kører når app'en starter
ViewLocator.cs - Kobler ViewModels til Views (MVVM “finder”)
Assets/ - Ikoner og statiske filer til UI
avalonia-logo.ico - Programikon
Views/ - UI vinduer (det brugeren ser)
LoginWindow.axaml - Login-skærm (brugernavn/password)
MainWindow.axaml - Hovedvindue (ordrer/robot/status)
RegisterWindow.axaml - Opret bruger (registration/admin)
ViewModels/ - UI-logik (knapper/commands + data til UI)
ViewModelBase.cs - Baseklasse: UI kan opdatere sig automatisk
MainWindowViewModel.cs - “Hjernen” bag MainWindow (DB + robot flow)
Models/ - Programmerings-objekter for domæne + robot
Domain.cs - Domæne-data: Order, OrderLine, Inventory osv.
Robot.cs - TCP/IP kommunikation til robot (send URScript)
RobotPositions.cs - Poses + URScript sekvenser (A0/A10 + gripper/sensor)
Data/ - Database-lag (SQLite + EF Core)
InventoryDbContext.cs - DB definition: tabeller/relations i inventory DB
Entities.cs - “DB rækker” (EF Core entities)
EntityMapper.cs - Oversætter Entities <-> Domain modeller
DbSeeder.cs - Fylder DB med testdata / reset
DbReader.cs - Læser DB data til UI (orders/status)
Auth/ - Login & brugere (security/authentication)
AuthDbContext.cs - DB for brugere/login
Account.cs - Bruger-objekt (username + password hash)
AccountService.cs - Login-logik (hash/validering/opret bruger)
Session.cs - Hvem er logget ind lige nu
README.md - Kort guide til programmet
Class_Diagram_Inventory_System.png - Dokumentation