

Maschinenverwaltung

Objektorientierte Programmierung 2

Höhere Fachschule für Technik Mittelland



Grenchen, 20.02.2023
Patrick Schreyer, Yannic Ziegler
Version 1.0.0

Inhaltsverzeichnis

Erstellung	3
Glossar	3
Szenarienbeschrieb	5
Digitalisierung	5
Grundlage	6
Ausgangslage	6
Ziel der Applikation.....	7
Rahmen.....	7
Vorgehen	8
Wahl des Szenarios	8
Planung	9
Abgrenzung.....	10
Entwicklungsumgebung.....	10
Anforderungen	11
Ziel-Definition (Scope)	11
Business-Use-Cases.....	12
Programm Use-Cases.....	13
Wireframe.....	14
Klassendiagramm.....	15
Persistenz.....	16
Code Dokumentation	17
Genereller Aufbau	17
Unit Tests	18
Enumeration	19
Vererbung	19
Casting	19
Interfaces	20
abstrakte Klasse	20
Generics	20
Collections und Sortierung.....	21
Java Serialisierung.....	23
Exceptions	24
Threads	24
Logfiles	25
Learnings	26
User Experience	26
Entwicklung Wireframe ➡ App.....	27
Speichern von Inhalten	28
Starten der Applikation.....	28
Benutzerhandbuch	29
Kunden-Ansicht.....	29
Hersteller-Ansicht	32

Erstellung

Patrick Schreyer
Servicetechniker
AxNum AG

Softwareentwicklung
Klasse BBIN21.1c

Yannic Ziegler
Innovation Engineer
ETA SA Manufacture Horlogère Suisse

Wirtschaftsinformatik
Klasse BBIN21.1c

Glossar

Für das Gemeinsame Verständnis unserer App «Maschinenverwaltung» erstellten wir sehr früh ein Glossar, welches die wichtigsten Begriffe und deren Verwendung definiert.

Zum Verständnis der nachfolgenden Kapitel ist das Glossar gleich hier abgebildet.

Term (english)	Begriff (deutsch)	Beschreibung oder Definition (english / deutsch)
Enterprise	Unternehmen	Unternehmen welches die «Maschinenverwaltung» einsetzt.
Area	Anlagenkomplex	In der «Maschinenverwaltung» pflegbares Objekt welches mit: <ul style="list-style-type: none">• definierbarem logischen Anlagenkomplex gemäss interner Strukturen• einem Stockwerk• einem Werk• einer unabhängigen SCADA Einheit organisatorisch einer verantwortlichen Person
Process Cell ¹	Anlage	In der «Maschinenverwaltung» pflegbares Objekt welches eine physische Anlage mit eindeutiger Seriennummer repräsentiert.
Supplier	Hersteller	Hersteller einer Anlage welche in der «Maschinenverwaltung» gepflegt sind.
Customer	Kunde	Kunde von Anlagen welche in der «Maschinenverwaltung» gepflegt sind. Kann selbst Anlagenkomplex erstellen und die eigenen Maschinen diesen zuordnen.
Site	Werk / Unternehmens-Standort	Standort einer Produktionsstätte eines Unternehmens mit definierter Adresse und eindeutiger Bezeichnung zur Verwaltung im eingesetzten ERP System.
Floor	Stockwerk / Geschoss	Name oder Nummer der Etage, in der sich dieser Anlagenkomplex befindet.

¹ Gemäss ISA-88

Manager	Manager	Manager:in eines Anlagenkomplex.
ERP	-	Enterprise Ressource Planning System, um Ressourcen des Gesamtunternehmen zu steuern (Finanzen, Lager ...).
MES	-	Manufacturing Execution System System, um die Produktion in einem Unternehmen zu steuern.
SCADA	-	Supervisory Control and Data Acquisition System, um mehrere Maschinen eines vernetzten Bereiches intelligent basierend auf erfassten Daten zu steuern.
HMI	-	Human Machine Interface Schnittstelle (meist Display/Buttons) von Mensch zu Maschine.
PLC	-	Programmable Logic Controller Steuereinheit welche die Logik der Maschine enthält und sie steuert.

Die Klassen im Programm richten sich nach der englischen Bezeichnung, wobei das User Interface auch die deutschen Übersetzungen enthalten kann.

Szenarienbeschrieb

Industrie 4.0, Internet of Things, Digitalisierung, Smart Factory, digitale Transformation ...

Dies sind nur einige «Buzz-words», welche viral gehen und omnipräsent auf LinkedIn, an jeder Messe und in allen Industrie-Zeitschriften anzutreffen sind.

Übersetzt heisst das, dass sich heutige Industrieanlagen einerseits an rechtliche Maschinennormen halten müssen und zusätzlich viele Industrienormen und Standards erfüllen sollten, damit Steuerungen interoperabel mit anderen Maschinen kommunizieren können.

Digitalisierung

Das Thema Digitalisierung, insbesondere die digitale Transformation, dreht sich um den Menschen und wie dieser mit der Technologie umgeht. Gute digitale Lösungen halten sich an Standards, sind vernetzbar, generieren Mehrwert für das Unternehmen und Erfüllen unterbewusste menschliche Anforderungen an Softwaresysteme (User Experience).

Bereits in kleinen Unternehmen ist dies eine Challenge, denn Maschinen werden auf ein Bedürfnis gekauft und dann in die Fertigungshallen, welche international verteilt sein können, integriert. Bald werden sie vernetzt, rapportieren Daten und tragen enorm zur Wertschöpfung bei. Schon ein kurzer Ausfall einer Anlage kann immense finanziellen Folgen mit sich ziehen. Ein möglichst reibungsloser Betrieb ist somit für alle Stakeholder erstrebenswert.



Abbildung 1 Produkte zum Beschriften von AxNum

Grundlage

Für diese Arbeit setzen wir auf industrielle Standards. Relevant ist in unserem Sektor besonders die EU Maschinenrichtline. Sie definiert, was eine Maschine ist, und welche Richtlinien gelten (CE-Thematik).

Ergänzend dienen die ISA-88 und ISA-95 Normen, worauf die Architektur und Benennung der Klassen in unserer App «Maschinenverwaltung» beruht.

EU RICHTLINIE 2006/42/EG	Maschinen-Richtlinie Die Norm regelt ein einheitliches Schutzniveau zur Unfallverhütung für Maschinen und unvollständige Maschinen beim Inverkehrbringen innerhalb des Europäischer Wirtschaftsraum. Darüber hinaus hat die Schweiz die Regelungen der Maschinenrichtlinie größtenteils in nationales Recht übernommen.
ISA-88	Norm für die chargenorientierte Fahrweise Sie ist eine Designphilosophie für Software, Ausrüstung und den Verfahrensablauf.
ISA-95	Norm für die Integration von Unternehmens- und Betriebsleitebene Die ISA-95 basiert auf ISA-88, erweitert diese von der Prozessleittechnik für den Batchbetrieb auf die Betriebsleittechnik, die auch für diskrete und kontinuierliche Fertigung anwendbar ist.

Ausgangslage

Wir nehmen an, dass ETA SA Manufacture Horlogère Suisse eine gewisse Anzahl Beschriftungslaser sowie Pressen von AxNum gekauft hat. Diese ständen nun in mehreren Abteilungen produktiv im Einsatz.

Der Kunde wird diese **Anlagen [Level 0]** als Asset in seinem **ERP [Level 4]** erfassen um einerseits die Abschreibungen sowie Investitionen für die Finanzen zu hinterlegen. Dabei wird automatisch ein Werk definiert damit für die Versicherung bei einem Brand beispielsweise der Verlust belegt werden kann. Die Anlage wird dann meistens ans Netzwerk des Unternehmens angebunden um von den gesammelten Daten zu profitieren.

Moderne Maschinen sind vernetzt und kommunizieren untereinander. Die Daten zur Steuerung der gesamten Produktion werden im **MES [Level 3]** verwaltet. Einzelne Daten werden zurück ins ERP gespielt und sind beispielsweise zur Rückverfolgung der Kosten und Produktionsdaten relevant.

Ein Bereich, also ein Bereich mit mehreren vernetzten Maschinen, wird mit **SCADA [Level 2]** gesteuert. Dadurch kann reaktiv auf Störungen, Nachladen von Rohmaterial und die präzise Steuerung von Stückzahlen oder beispielsweise den Test-Lot eingegangen werden.

Jede Anlage wird schlussendlich von einer **PLC [Level 2]** gesteuert und verarbeitet unterschiedliche Signale der Sensoren zur Steuerung des Prozesses auf der Maschine.

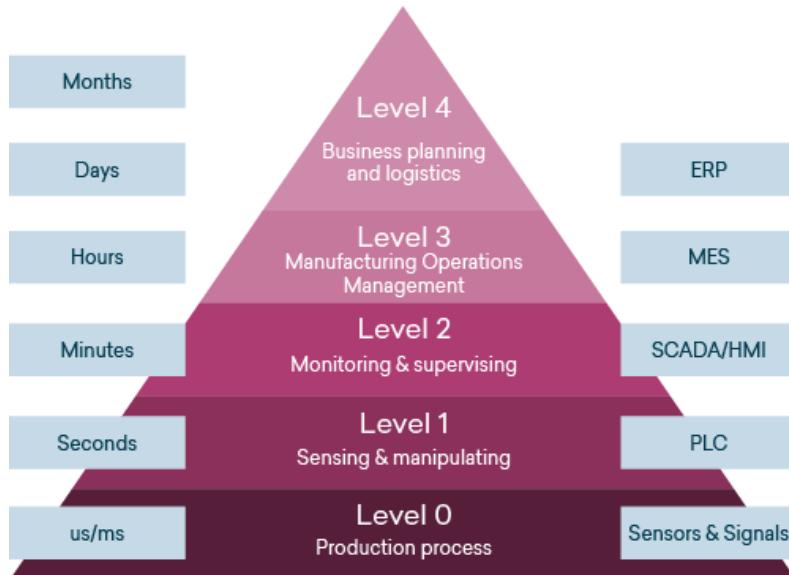


Abbildung 2 ISA-95 Daten-Pyramide mit welchen die «Maschinenverwaltung» zusammen agiert.

Ziel der Applikation

Nun erhalten Maschinen-Hersteller häufig Support- oder Unterhalts-Aufträge. Ein Stillstand kostet viel Geld, daher sollten diese schnellstmöglich die Maschine wieder zum funktionstüchtig machen können.

Damit die aufgebotenen Mechaniker wissen, wo sich die betroffene Maschine gerade befindet, stellt der Hersteller seinen Kunden die «Maschinenverwaltung» App zur Verfügung. In Grossunternehmen wird diese App an das bestehende ERP angeschlossen und erbt die erfassten Werk-Informationen.

Manuell können dann Areas (Anlagenkomplexe) geführt werden, um die Maschinen den richtigen Standorten zuzuweisen. Möglicherweise ist dafür bereits ein Modul wie bspw. ein «Enterprise Asset Management» (EAM) im betroffenen Unternehmen eingeführt. So könnten die Daten auch wieder mit der zentralen Datenbank synchronisiert werden und müssen nur an einem Ort unterhalten sein.

Rahmen

Der Scope unserer Arbeit begrenzt sich auf eine simple Verwaltung von **Anlagenkomplexen** und **Anlagen**, damit Hersteller ihren Kunden eine Plattform zur Zusammenarbeit anbieten können.

Im Kontext steht die künftige Möglichkeit das Tool für mehrere Unternehmen anzubieten.

Vorgehen

Dieses Kapitel beschreibt unser Vorgehen im Projekt und zeigt den Wissenstransfer aus anderen Modulen an der hftm auf.

Wahl des Szenarios

Wir beide kommen aus dem Maschinenbau und sind bei unterschiedlichen Unternehmen angestellt. Indirekt hatten wir bereits beruflich zusammen gearbeitet.

Wir stellten in einer Diskussion fest, dass dieselben Probleme im Sondermaschinenbau bestehen. Dieses Projekt soll uns dienen, Erfahrungen aus der täglichen Arbeit miteinzubeziehen, Entwicklungs-Erfahrungen zu sammeln und idealerweise ein Tool erstellen, dessen Konzept zurück in unsere Unternehmen fliessen kann.

Um das Ziel zu definieren, Abgrenzungen zu tätigen und uns abzusprechen, setzten wir auf den gelernten Problemlösungszyklus aus dem Requirements Engineering.

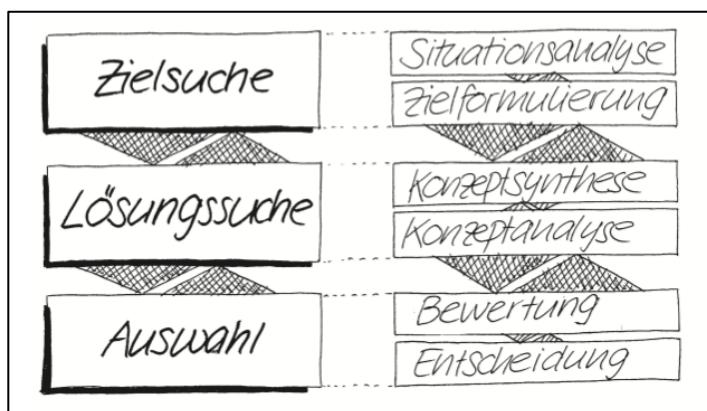


Abbildung 3 Problemlösungszyklus

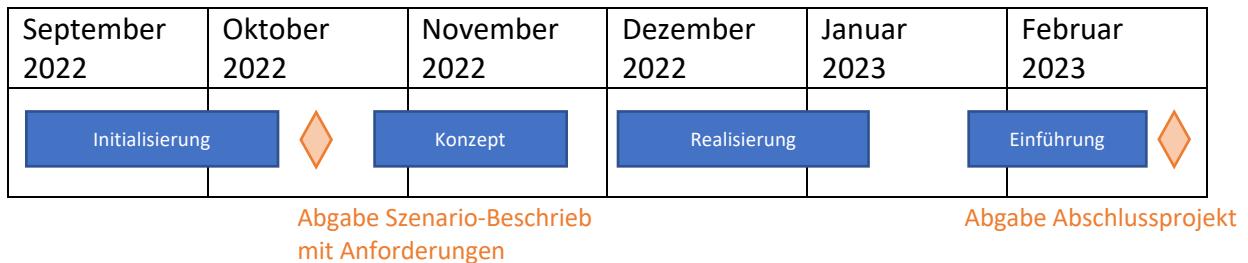
Wir haben dabei nicht alle Anforderungen explizit dokumentiert, sondern uns mit diversen Teams-Calls ein gemeinsames Verständnis (implizite Anforderungen) geschaffen.

Disclaimer

Dieser Teil der Dokumentation beschreibt nicht die Applikation selbst sondern den Prozess zum Endprodukt hin. Es sind inkonsistente Anforderungen in Diagrammen, Schemas oder Texten enthalten, welche für eine technische Dokumentation eines echten Produktes noch bereinigt werden müssten.

Planung

Wir wählten HERMES als Projektmanagement Methode mit den Standardphasen, aber setzten lediglich die zwei vorgegebenen Meilensteine ein:



Initialisierung

Sehr bald wussten wir welches Projekt wir verfolgen möchten. Nach einem gemeinsamen Gespräch über die Arbeit erstellte Yannic ein erster Entwurf des Szenarienbeschrieb und integrierte die Illustrationen der Industrienomen. Dies war die Grundlage für unsere erste Diskussion wie wir die App nun abgrenzen wollen.

Konzept

Nach Schaffung eines Gemeinsamen Verständnis zur Thematik, erstellten wir die ersten Schemas (Use-Case Diagramm, Klassendiagramm) und ein Wireframe für das User Interface während einem Konferenzcall.

Realisierung

Die initialen Klassen erstellte Yannic und setzte das Git Repository auf. Mit diesen Informationen programmierte Patrick die Views und Controller und implementierte den Grossteil der Applikation.

Einführung

Wir haben den Zwischenstand einige Wochen vor Abgabe gemeinsam besprochen und die weiteren Arbeiten (Testing, Bug-Behebung, Dokumentation ...) aufgeteilt.

Abgrenzung

Wir erstellen eine Java Applikation, welche einige Daten persistent führt. Dabei setzen wir auf die Industrienormen, welche die Datenerhebung strukturieren.

Mit Hilfe von Industrienormen grenzen wir ab, welche Daten im System editierbar sind (Scope), welche «hardcodiert» werden (Kontext) und welche für das System irrelevant sind.

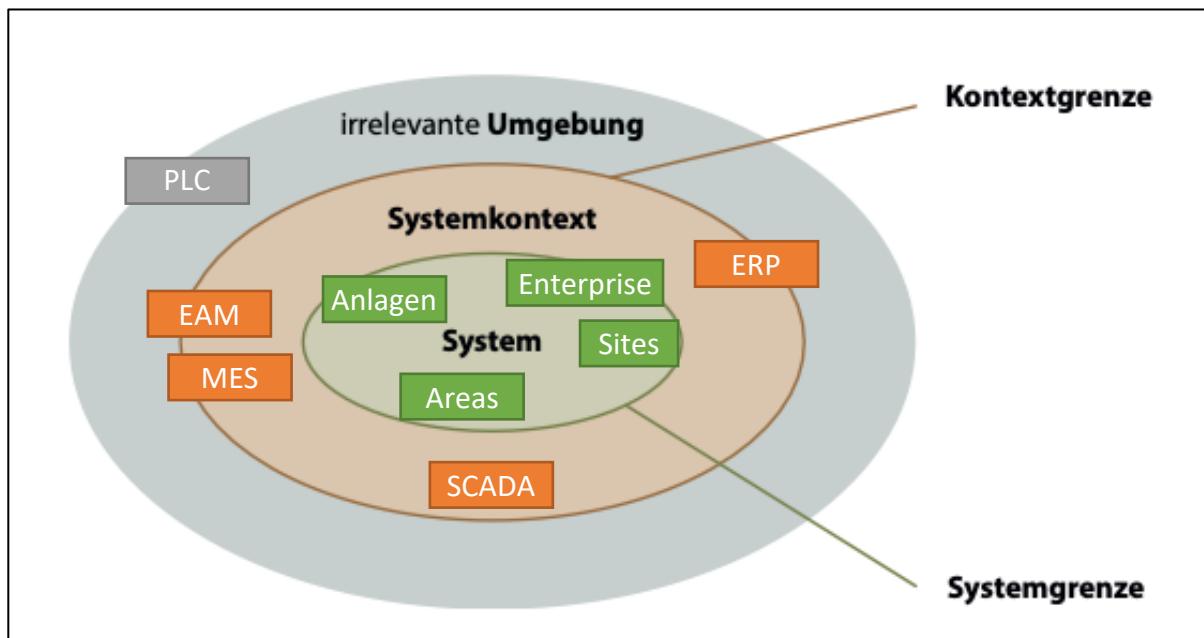


Abbildung 4 Requirements Engineering System, Kontext und irrelevante Umgebung

Entwicklungsumgebung

Wir entwickeln mit Java FX in der IDE Visual Studio Code. Dabei kommt für das Frontend der Scene Builder für fxml Dateien zum Einsatz.

Das Projekt steht auf **GitHub** bereit:

<https://github.com/SirZesso/Maschinenverwaltung>

Die Commits sind in einer History ersichtlich und zeigen die Aufteilung der Arbeiten auf.

Wichtig

Es handelt sich um ein privates repo und Berechtigungen müssen vergeben sein, um Zugriff zu erhalten.

Anforderungen

Dieses Kapitel dokumentiert die «nicht-sichtbaren» Arbeitspakete welche im Rahmen des Requirements Engineering im Projekt «Maschinenverwaltung» entstanden sind. Dazu gehört ebenfalls diese Dokumentation, das Readme auf GitHub und das enthaltene Glossar.

Ziel-Definition (Scope)

In unserem Programm möchten wir **Anlagen** (ProcessCell's) führen, welche sich in **Anlagenkomplexe** (Area) befinden und diverse Prozeduren abarbeiten.

Der Kunde der Software ist **ETA SA**, welche mehrere Werke (Site) führt. Einzelne Abteilungen fassen Anlagen (Process Cell) von **AxNum** zusammen und bilden einen Anlagenkomplex (Area).

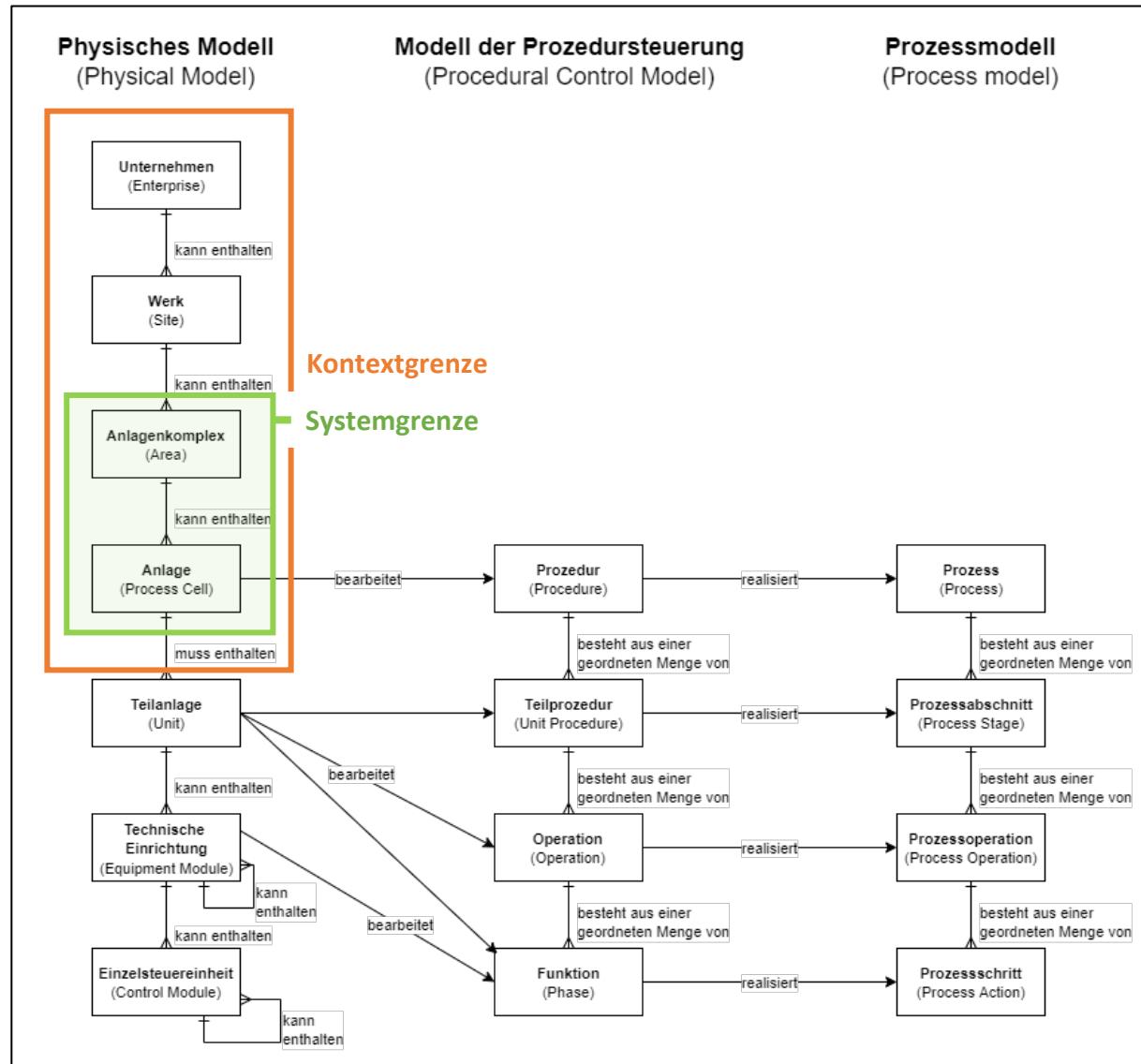


Abbildung 5 Industrienorm ISA-88, Maschinenaufbau

Business-Use-Cases

AxNum bietet ETA SA eine einfache Verwaltung ihrer Anlagen an. ETA kann zudem selbstständig Areas definieren und Maschinen zuweisen.

Dies ermöglicht eine saubere Dokumentation, Übersicht und gemeinsames Verständnis beider Geschäftspartner.

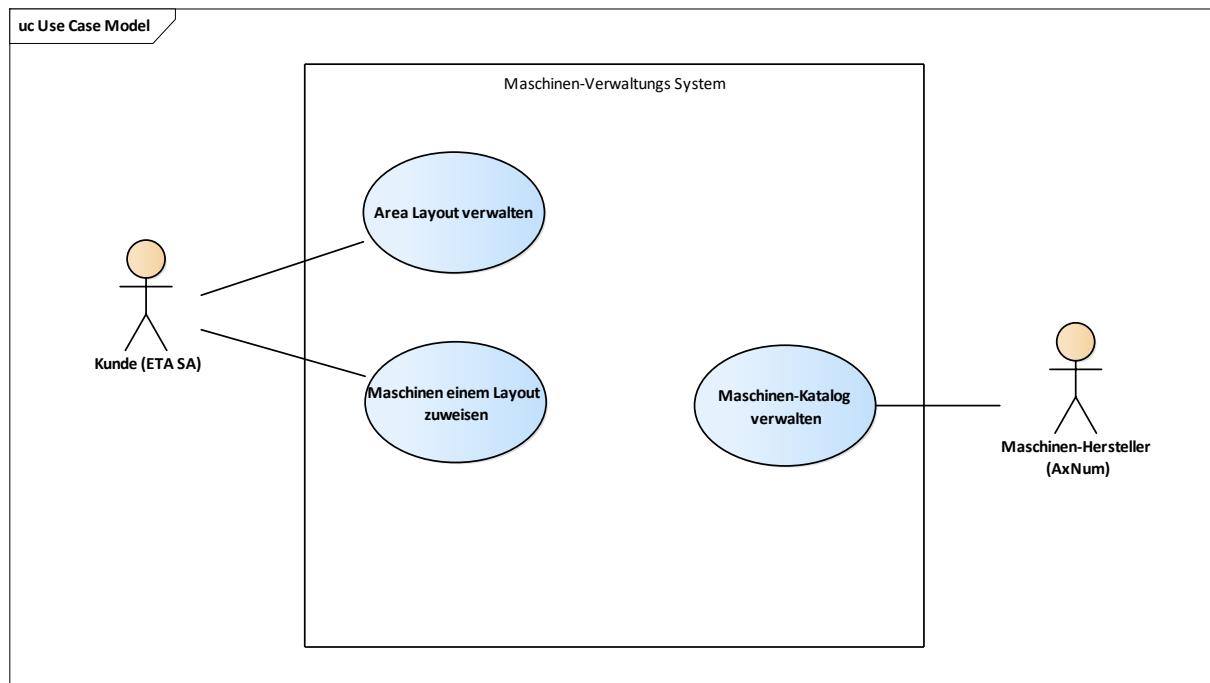


Abbildung 6 Use Case Diagramm

Programm Use-Cases

Hier sind zwei Beispiele von möglichen Einträgen in der «Maschinenverwaltung» mit den Links auf die Grundlage (öffentliche Website).

Hinweis

Im Projekt werden alles «**Dummy-Daten**» enthalten sein, welche nicht mit Zahlen oder Angaben der beiden Unternehmen übereinstimmen (müssen) 😊

Area	ProcessCell
	
<p>Produktions-Linien:</p> <ul style="list-style-type: none">• Verpressen von Unterbaugruppen• Beschriften von Uhrenschalen <p>https://www.eta.ch/de/wo-wir-sind</p> <p>Beispiel-Produktionslayouts (fiktiv)</p>	<p>Produkte:</p> <ul style="list-style-type: none">• Pressen• Laser <p>https://www.axnum.ch/de/pressen/produkte/ https://www.axnum.ch/de/beschriften-markieren/produkte/</p> <p>Nur Produkte, welche einer kompletten Maschine entsprechen.</p>

Wireframe

Wir können das User Interface in zwei Sichtweisen unterteilen. Beide Benutzer sind eng miteinander verknüpft und können modularisierte Screens nutzen.

Das Erscheinungsbild ist in die «Hersteller-Sicht» und «Kunden-Sicht» aufgeteilt und schafft so eine klare Trennung. Für die künftige Weiterentwicklung sollte der Fokus sicher auf dem Kunden-Bereich liegen, um ihm möglichst viel Mehrwert zu bieten.

Hinweis

Die einzelnen Sichten kann man in der Applikation von einem «Login-Screen» den Bereich zu Demozwecken aufrufen.

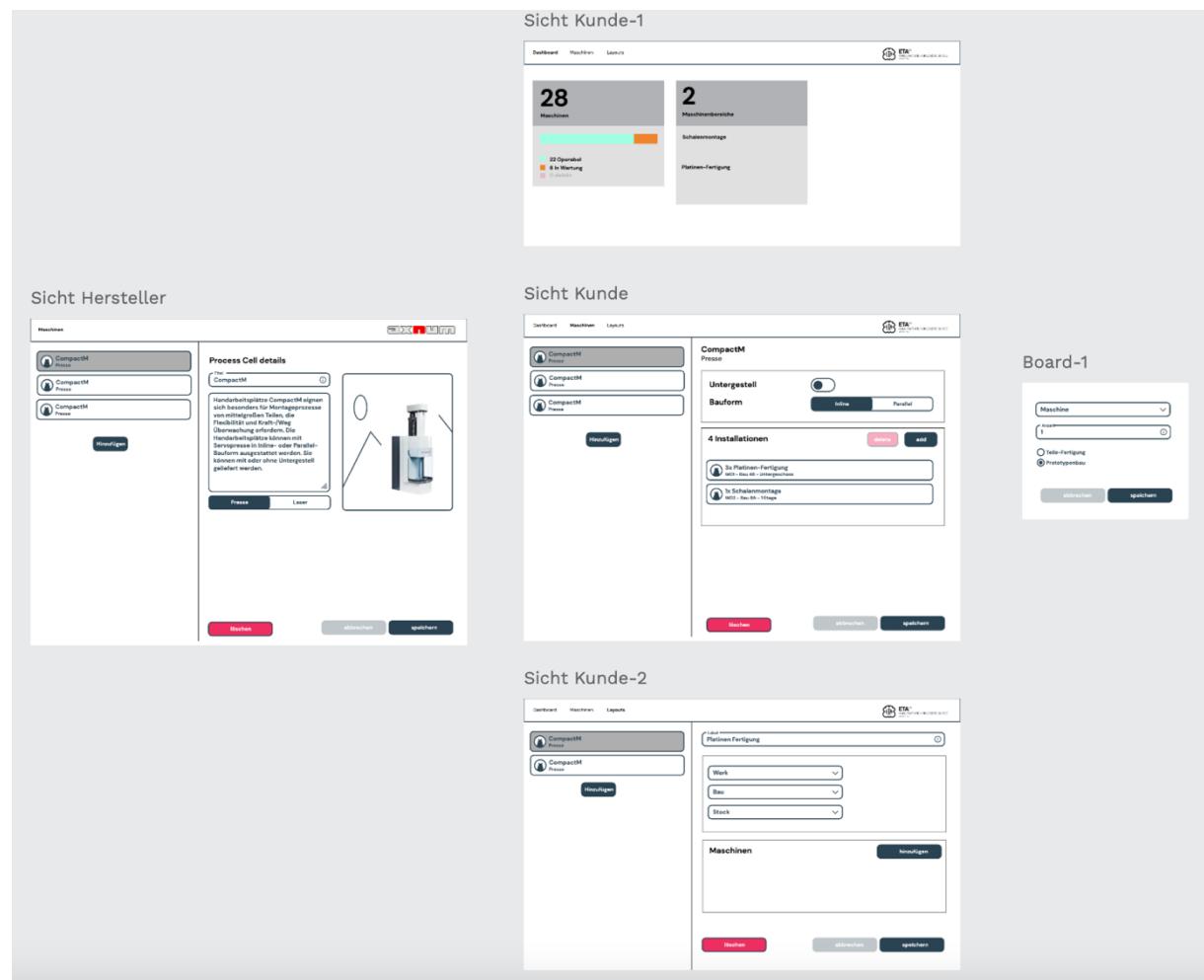


Abbildung 7: Wireframe Entwurf erstellt mit Penpot, vor Realisierung der App.

https://design.penpot.app/#/workspace/69cd8e21-398c-8036-8001-8cf2201211dd/69cd8e21-398c-8036-8001-8ad3c839b5c4?page_id=69cd8e21-398c-8036-8001-8ad3c839b5c5

Klassendiagramm

Das erstellte Klassendiagramm hilft uns zur Aufgabenaufteilung und Visualisierung der Schnittstellen für das Programm.

Hinweis

Die grauen Objekte sind in der «Maschinenverwaltung» nur im Code anpassbar, werden aber bereits lokal gespeichert und sind einfach austauschbar. Die anderen Objekte können im GUI verwaltet werden.

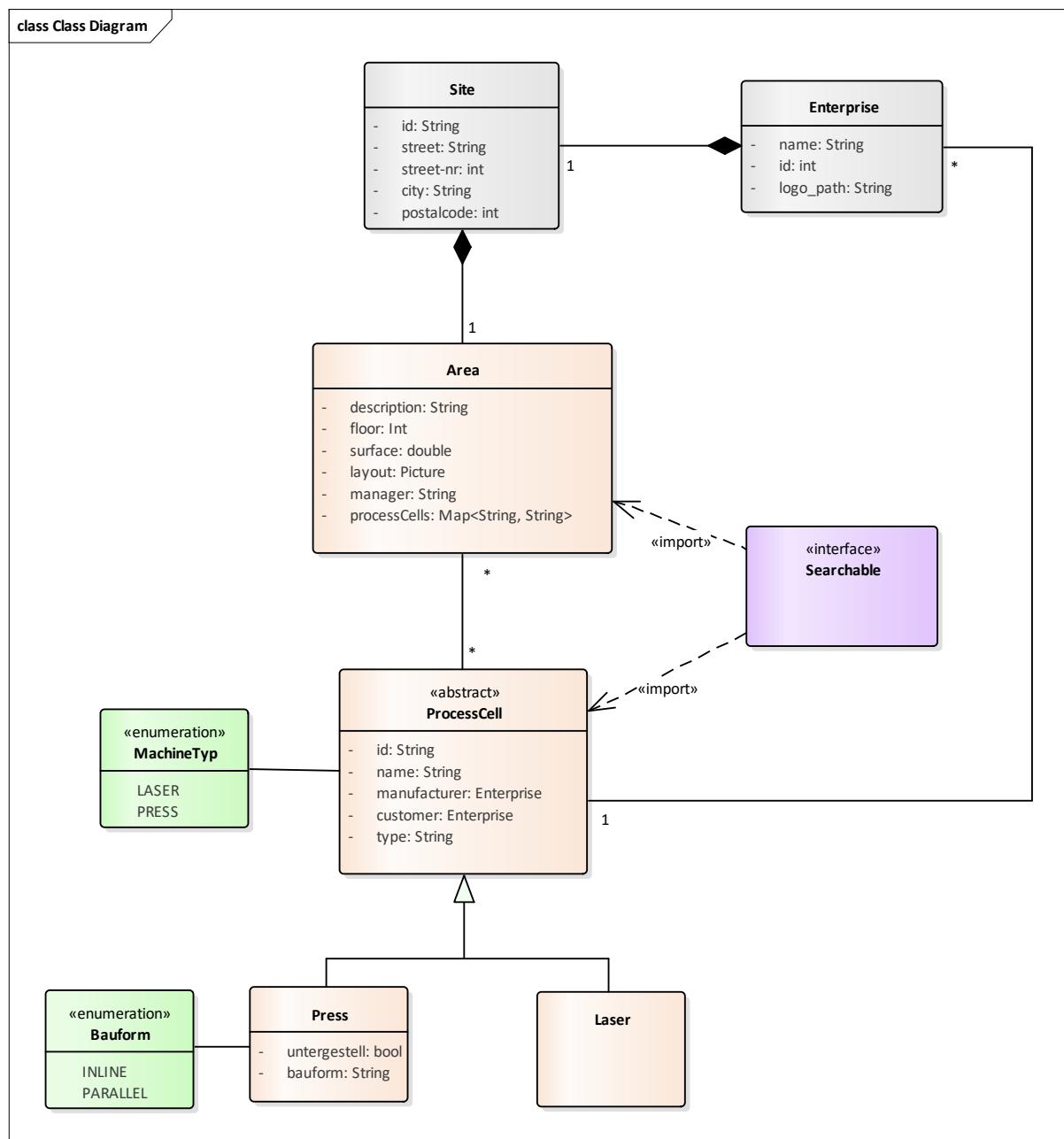


Abbildung 8: Klassendiagramm erstellt in Enterprise Architect im Stand vor der Realisierung der Applikation

Persistenz

Wir haben uns diverse Gedanken zur Persistenz gemacht und zwei Optionen zur Speicherung der Daten erarbeitet:

- A) Anbindung einer Datenbank
- B) Lokale Serialisierung

Eine normalisierte Datenbank haben wir vorbereitet, jedoch müssten wir dann alles noch auf einem Server hosten. Da dies nicht Teil dieses Projektes ist, haben wir uns entschieden die gelernte lokale Serialisierung anzuwenden und einfach in lokale Files zu speichern.

Option A: Anbindung einer Datenbank (Skizzen während der Planung):

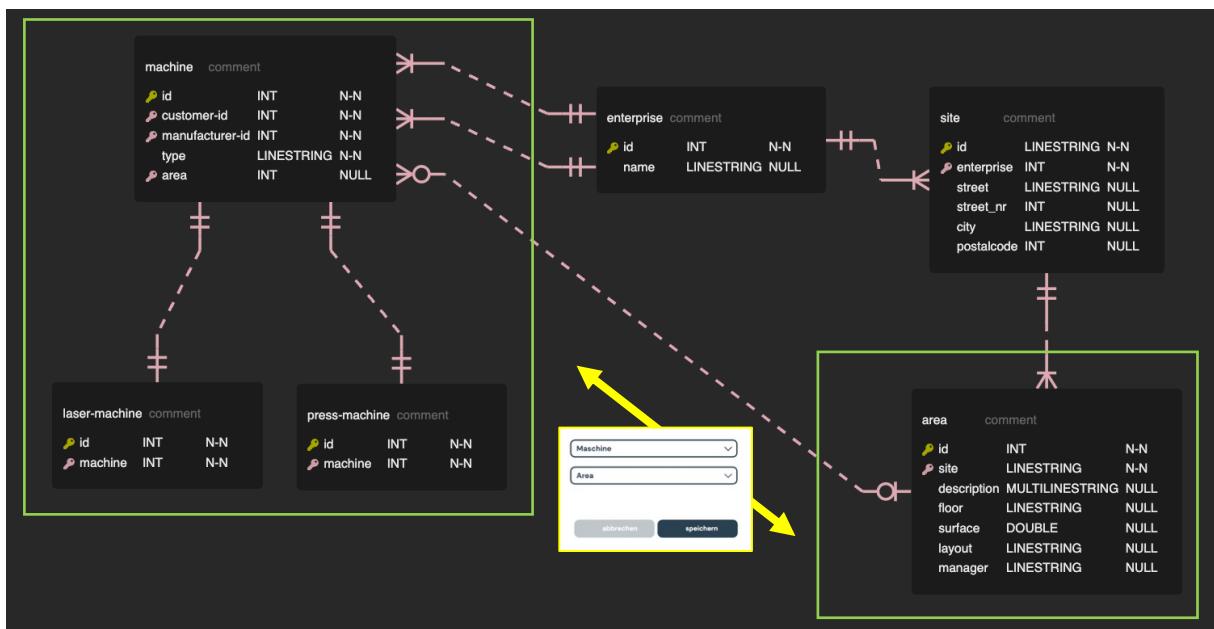


Abbildung 9: Datenbankschema erstellt in VS Code mit VUERD

Eine Process Cell (vom Typ Laser oder Presse) steht dabei auf genau einem Area, welches einer Site zugeordnet ist. Sie wird von einem Hersteller (Enterprise) produziert und gehört einem Kunden (Enterprise).

Unternehmen

	<input type="button" value="T"/>	<input type="button" value="A"/>	<input type="button" value="id"/>	<input type="button" value="name"/>
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	1 AxNum
<input checked="" type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	2 ETA SA

Sites

	<input type="button" value="T"/>	<input type="button" value="A"/>	<input type="button" value="id"/>	<input type="button" value="enterprise"/>	<input type="button" value="street"/>	<input type="button" value="street_nr"/>	<input type="button" value="city"/>	<input type="button" value="postalcode"/>
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	1	Schild-Ruststrasse	16	Grenchen	2540

Hinweis

Die realisierte Option B ist in der Code Dokumentation abgebildet.

Code Dokumentation

Die Code Dokumentation dient zur Übersicht unseres Programmes. Mit ihr soll es einem neuen Entwickler möglich sein rasch im Projekt zurechtzukommen. Screenshots wurden eingebunden um Beispiele zur Bewertung unserer Arbeit gleich zu referenzieren.

Genereller Aufbau

Wir haben uns an das MVC-Pattern bei der Erstellung der Applikation bestmöglich gehalten:

<pre>▽ MASCHINENVERWALTUNG ▽ src / main ▽ java ▽ controller J CustomerController.java J EnterpriseController.java J LoginController.java J ProcessCellController.java J SupplierController.java ▽ main J AreaPane.java J launch.java J MachineView.java J MainApp.java J NumberTextField.java J SharedMenu.java ▽ model > customer > machine J Enterprise.java J Overview.java ▽ service J SerializationService.java ▷ resources > target ◇ .gitignore ▷ pom.xml ① README.md</pre>	<p>Das Modell (model) Es enthält die Arbeitsdaten des Programms wie Objekt-Klassen, Nutzereingaben, aus Datenbanken gelesene Daten, etc. Es enthält niemals Referenzen auf die beiden anderen Teile.</p>
	<p>Die Steuerung (controller) Die vermittelnde Schicht, die für die Interaktion zwischen Präsentationsschicht und Daten zuständig ist. Sie wird oft mittels eines Observer-Patterns realisiert, das die Daten des Modells beobachtet und diese ggf. mit der Präsentationsschicht wechselseitig aktualisiert. Die Steuerung muss somit sowohl zu den Daten, als auch zu Teilen der Darstellungsschicht Zugang haben.</p>
	<p>Die Präsentation (view) Die (bei einem Desktop-Programm z.B. graphische) Darstellung von Daten, die Programmoberfläche (GUI), etc. Sie enthält im Quelltext weder Daten, noch Teile der Geschäfts- oder Programmlogik vor, sondern ist lediglich für deren Darstellung verantwortlich. Hierzu definiert sie entsprechende Schnittstellen.</p>
	<p>Der Service (service) Enthält den Serealisierungs-Service um diese geteilte Logik von den Controller zu separieren.</p>
<pre>▽ resources ▽ docs Ȉ database_setup.sql Ȉ database.vuerd > images ▽ serialisation Ȉ areas.ser Ȉ customers.ser Ȉ enterprises.ser Ȉ manufactures.ser Ȉ processCells.ser ▽ styles # Styles.css ▽ view ▽ customer Ȉ assigneProcessCeltoArea.fxml Ȉ layouts.fxml ▽ supplier Ȉ layouts.fxml Ȉ login.fxml</pre>	<p>In den Ressourcen werden die Bilder vom Programm sowie die serealisierten Daten der Objekte abgespeichert.</p>
	<p>Die Views sind als fxml systembedingt hier vorzufinden.</p>

Unit Tests

Die Unit Tests haben wir bei den wichtigsten Klassen Area, Laser und Press erstellt. Diese Elemente werden miteinander verknüpft und wir müssen prüfen, ob sie auch richtig erstellt wurden und vorhanden sind.

Laser Test

Die beiden Tests für die ProcessCell sind sehr rudimentär gehalten. Ein Objekt wird erstellt und dann das Attribut «Wellenlänge» des Lasers geprüft, ob es dem getesteten Wert entspricht (Getter + Setter).

```
① 8  public class LaserTest {
 9
10
11    @Test
12    public void testGetWavelength() {
13      Laser laser = new Laser(serialnumber: 1, name: "TestLaser", manufacturer: null, customer: null, type: null,
14      assertEquals(expected: 100, laser.getWavelength());
15    }
16
17    @Test
18    public void testSetWavelength() {
19      Laser laser = new Laser(serialnumber: 1, name: "TestLaser", manufacturer: null, customer: null, type: null,
20      laser.setWavelength(wavelength: 200);
21      assertEquals(expected: 200, laser.getWavelength());
22    }
23
24    @Test
25    public void testWavelengthProperty() {
26      Laser laser = new Laser(serialnumber: 1, name: "TestLaser", manufacturer: null, customer: null, type: null,
27      assertEquals(expected: 100, laser.wavelengthProperty().get());
28      laser.wavelengthProperty().set(value: 200);
29      assertEquals(expected: 200, laser.wavelengthProperty().get());
30    }
}
```

Press Test

Gleichermassen ist die Presse aufgebaut; dort testen wir das andere Attribut «Newton».

Area Test

In der Area haben wir uns entschieden diverse Attribute zu testen. Dabei ist die Erstellung in einer Funktion «setUp()» zusammengefasst. Damit das mvn package erstellt werden konnte, musste trotz der Annotation @BeforeEach bei zwei Tests die Funktion direkt aufgerufen werden, sonst war ein Paketieren nicht möglich.

```
J AreaTest.java ×
src > test > java > model > customer > J AreaTest.java > ⓘ AreaTest > Ⓜ testSetSiteId()
1  package model.customer;
2
3  import org.junit.jupiter.api.Assertions;
4  import org.junit.jupiter.api.BeforeEach;
5  import org.junit.jupiter.api.Test;
6
7  public class AreaTest {
8    private Area area;
9
10   @BeforeEach
11   public void setUp() {
12     area = new Area(siteId: "1", name: "Area1", description: "This is Area1", Floor.UG2, surface: 100.0, manager: "John");
13   }
14
15   @Test
16   public void testGetSiteId() {
17     setUp();
18     Assertions.assertEquals(expected: "1", area.getSiteId());
19   }
20
21   @Test
22   public void testSetSiteId() {
23     area.setSiteId(siteId: "2");
24     Assertions.assertEquals(expected: "2", area.getSiteId());
}
```

Enumeration

Der Benutzer kann bei der Erstellung einer Anlage vom Typ Laser oder Presse den Maschinentyp wählen. Dabei unterscheiden wir manuelle Anlagen, welche von Menschen bedient werden und automatische Anlagen.

Beim Stockwerk haben wir uns ebenfalls für eine Enumeration entschieden. Auch wenn dies nachträglich nicht der idealste Weg ist (denn nicht alle Gebäude haben dieselbe Anzahl Stockwerke ...) muss es für unseren MVP reichen 😊.

```
src > main > java > model > customer > J Floor.java > ...
1 package model.customer;
2
3 public enum Floor {
4     UG2,
5     UG1,
6     EG,
7     OG1,
8     OG2,
9     OG3,
10    OG4
11 }
```

Vererbung

Bei den Vererbungen ist unser Objekt Process Cell das Beispiel. Wir haben aktuell zwei Anlagen-Arten integriert: Laser und Pressen. Beide haben 90% dieselben Attribute und erben diese von der Process Cell. Nur ein Attribut (Wellenlänge bei Laser und Newton bei Presse) wird Artenspezifisch angepasst:

```
public class Press extends ProcessCell{
    private IntegerProperty newton;
}
public class Laser extends ProcessCell{
    private IntegerProperty wavelength;
}
```

Casting

Wir casten beispielsweise die Presse und Laser Objekte

```
selectedProcessCell.setType(choiceboxType.getValue());
if (selectedProcessCell instanceof Press) {
    Press selectPress = new Press();
    selectPress = (Press) selectedProcessCell;
    selectPress.setNewton(Integer.parseInt(textfieldSpecial.getText()));
}
if (selectedProcessCell instanceof Laser) {
    Laser selectLaser = new Laser();
    selectLaser = (Laser) selectedProcessCell;
    selectLaser.setWavelength(Integer.parseInt(textfieldSpecial.getText()));
}
```

Interfaces

Als Interface setzen wir das Externalizable ein, welches bei der Serialisierung hilft.

```
implements Externalizable {
```

Wenn das allerdings eingesetzt wird, muss man den Out-und Input selbst implementieren:

```
@Override  
public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {  
    setSerialnumber(in.readInt());  
    setName((String) in.readObject());  
    setManufacturer((Enterprise) in.readObject());  
    setCustomer((Enterprise) in.readObject());  
    setType((MachineType) in.readObject());  
    setImagePath((String) in.readObject());  
    setArea((Area) in.readObject());  
    setWavelength(in.readInt());  
}
```

abstrakte Klasse

Im Frontend muss der Benutzer einen Maschinen-Typ bei der Erstellung einer neuen Maschien auswählen. Er kann dabei zwischen Laser oder Presse wählen. Daher wird die Klasse *ProcessCell.java* abstrakt geführt.

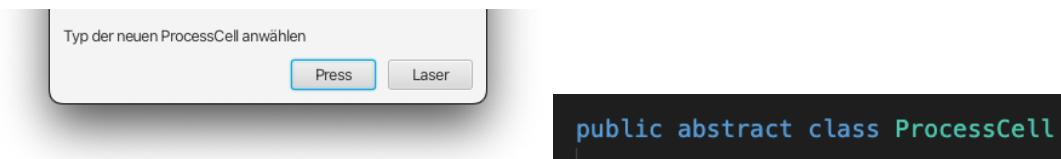


Abbildung 10 Auswahl Press / Laser zur Erstellung eines Objektes basierend auf einer abstrakten Klasse

Generics

Wir haben keine Generics selbst erstellt, aber diverse angewendet. Als Beispiel die ObjectProperty:

```
private ObjectProperty<Floor> floor;
```

Collections und Sortierung

Beim Laden der Areas ist ein schönes Beispiel eines Lamda Ausdruckes integriert, welches zusammen mit Streams und Collections filtert:

```
private void loadAreas() {
    areas.clear();
    // Add Areas from ProcessCells
    for (ProcessCell processCell : processCells) {
        if (!areas.contains(processCell.getArea())) {
            areas.add(processCell.getArea());
        }
    }
    // Add Areas from serialization
    List<Area> savedAreas = SerializationService.deSerializeAreaData();
    areas.addAll(savedAreas
        .stream().filter(
            area -> area.getName() != null && !areas.stream()
                .anyMatch(existingArea -> existingArea.getName() != null
                    && existingArea.getName().equals(area.getName())));
        .collect(Collectors.toList()));
}
```

Logik:

1. Rausladen aus ProcessCells
2. Areas deserialisiert
3. Mit **Streams** alle zugewiesen und gefiltert

User Interface:

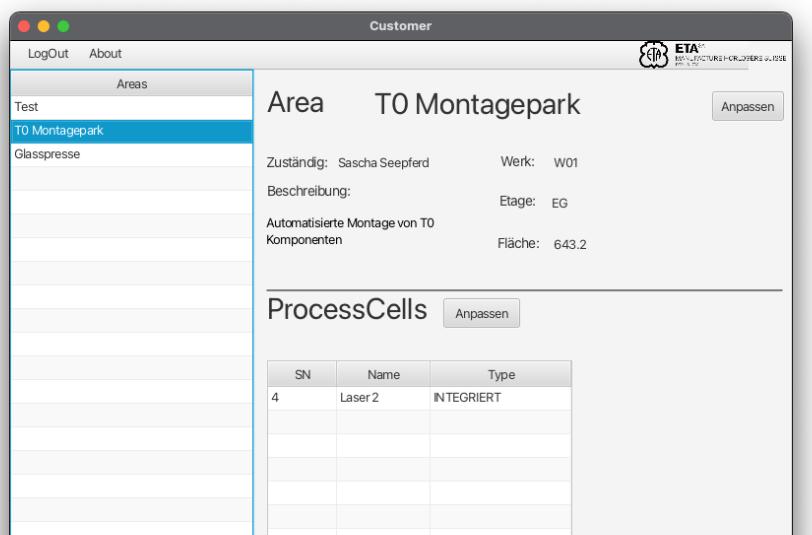


Abbildung 11 So hat übrigens das UI vor den Optimierungen ausgesehen.

Ebenfalls bei der Funktion setTables lassen wir die ProcessCells einfach von der einen in die andere Tabelle wechseln:

```
public void setTables() {
    assignedProcessCells = processCells.filtered(processCell -> {
        if (selectedArea != null && processCell.getArea() != null && processCell.getArea().getName() != null)
            return selectedArea.getName().equals(processCell.getArea().getName());
    })
    return false;
};

tabelAssignedProcessCells.setItems(assignedProcessCells);
columnAssignedNr.setCellValueFactory(cellData -> cellData.getValue().serialnumberProperty().asString());
columnAssignedName.setCellValueFactory(cellData -> cellData.getValue().nameProperty());
columnAssignedType.setCellValueFactory(cellData -> cellData.getValue().typeProperty().asString());

aviableProcessCells = processCells.filtered(processCells -> processCells.getArea().getName() == null);
tabelAviableProcessCells.setItems(aviableProcessCells);
columnAviableNr.setCellValueFactory(cellData -> cellData.getValue().serialnumberProperty().asString());
columnAviableName.setCellValueFactory(cellData -> cellData.getValue().nameProperty());
columnAviableType.setCellValueFactory(cellData -> cellData.getValue().typeProperty().asString());

}
```

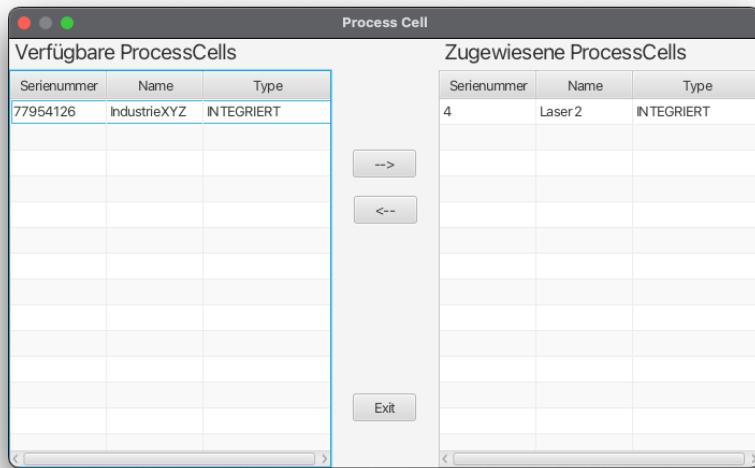


Abbildung 12 Feld zur Zuweisung von Laser und Pressen (Process Cells)

Java Serialisierung

Wie erwähnt setzen wir auf die Serialisierung und Deserialisierung, um unsere Daten persistent zu speichern. Die Files werden unter src/main/resources/serialisation in einzelnen files gespeichert.

Kunden werden von Hersteller getrennt, auch wenn es sich ums selbe Objekt «Enterprise» handelt. Dies machen wir absichtlich, um die beiden Sichten besser trennen zu können.



```
/*
 * Beispielhafte Serialisierung einer ObservableList
 */
public class SerializationService {

    //private final static String FILE_PATH = "src/main/resources/serialisation/";
    private final static String PROCESSCELL_PATH = "src/main/resources/serialisation/processCells.ser";
    private final static String ENTERPRISE_PATH = "src/main/resources/serialisation/";
    private final static String AREA_PATH = "src/main/resources/serialisation/areas.ser";

    //ProcessCell*****
    public static void serializeProcessCellData(ObservableList<ProcessCell> processCells) {
        try (FileOutputStream fileOut = new FileOutputStream(PROCESSCELL_PATH);
             ObjectOutputStream out = new ObjectOutputStream(fileOut)) {
            out.writeObject(new ArrayList<ProcessCell>(processCells));
        } catch (IOException e) {
            System.out.println(
                "ProcessCells: Aktuelle Daten konnten nicht fuer naechste Verwendung gespeichert werden:");
        }
    }
}
```

Die Demo-Daten erstellten wir in den jeweiligen Controller. Dabei sind die Methoden zur Erstellung einer «frischen» Demo-Datei auskommentiert:

```
@FXML
private void initialize() {

    // manufacturers = createDemoManufacturers();
    // customers = createDemoCustomers();
    // processCells = createDemoProcessCells();
    manufacturers = SerializationService.deSerializeEnterpriseDatao(file: "manufactures.ser");
    customers = SerializationService.deSerializeEnterpriseDatao(file: "customers.ser");
    processCells = SerializationService.deSerializeProcessCellDatao();
```

Exceptions

Wir setzen diverse Exceptions ein.

Im Serialization Service ein Beispiel einer IOException:

```
} catch (IOException e) {
    System.out.println(
        "Enterprises: Aktuelle Daten konnten nicht furr naechste Verwendung gespeichert werden:
    }
```

Ein sichtbares Beispiel ist die Methode isValid(), welche im Frontend die falschen Felder auch gleich rot einfärbt und einen Alert Dialog mit den Fehler erscheinen lässt.

```
private boolean isValid() {
    String errorMessage = "";
    boolean valid = true;
    if (textfieldAreaName.getText().isEmpty()) {
        errorMessage += "Name darf nicht leer sein!\n";
        textfieldAreaName.setStyle(value: "-fx-background-color: #ffad99");
        valid = false;
    }
    if (textfieldManager.getText().isEmpty()) {
        errorMessage += "Es muss jemand zuständig sein.\n";
        textfieldManager.setStyle(value: "-fx-background-color: #ffad99");
        valid = false;
    }
}
```

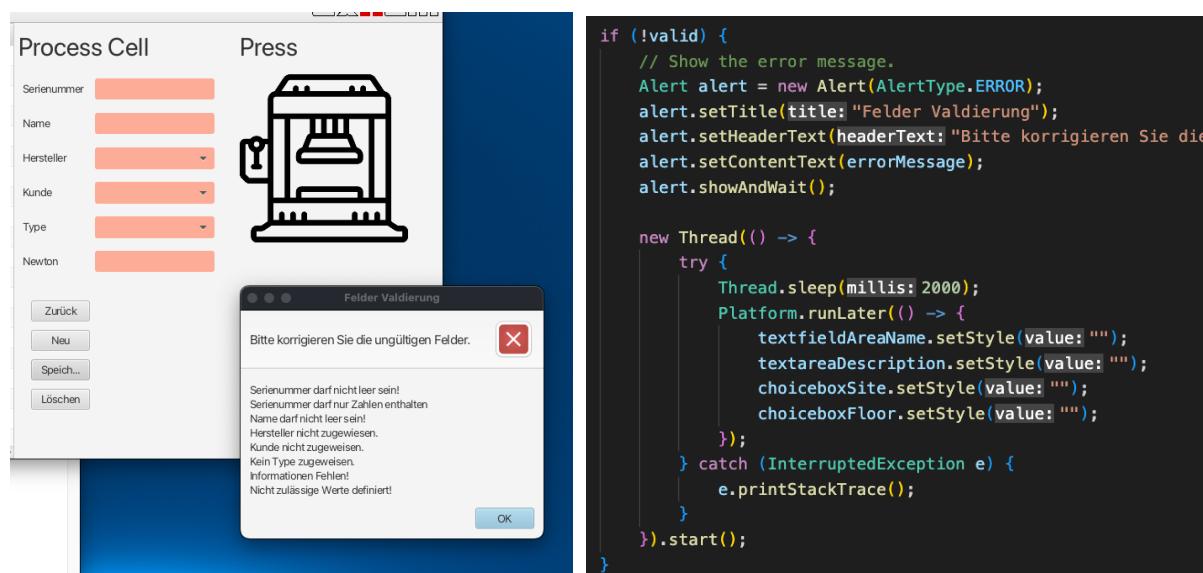


Abbildung 13 Ungültige Fehler, welche dank einem Thread rot erscheinen

Threads

Dies ist auch ein Bereich, wo wir einen neuen Thread erstellt haben. Er ist dafür zuständig, dass die Felder sich rot einfärben und die Fehlermeldung gleich Kontext verschafft. Dies ist hilfreich, wenn blass ein Feld falsch ist und man dadurch gar nicht suchen gehen muss wie das nun genau heisst.

Logfiles

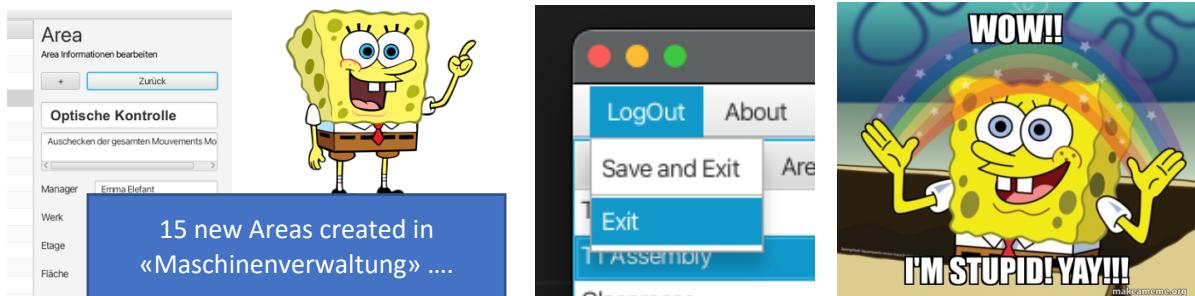


Abbildung 14 Logfile-Story in Meme-Format

Wir haben bemerkt, dass das User Experience (UX) gerade bei der Schaltfläche «Save and Exit» nicht optimal ausgearbeitet wurde... Schnell kann es sein, dass man sich verklickt und die gesamten Änderungen nicht gespeichert werden, weil man ausversehen «Exit» gewählt hat.

Wir könnten einen Alert zeigen und den Benutzer bei «Exit» auf sein Verhalten hinweisen...

Nun wollen wir aber diese Annahme erstmals prüfen, ob wirklich der grössere Teil der Nutzer dies als lästig empfinden könnte 😅. Darum verbauen wir nun einen Logger, welcher so Daten für statistische Auswertungen aufzeichnet, um unsere Annahme zu validieren oder zu widerlegen.

```
@FXML  
void btnClickExit(ActionEvent event) {  
    System.out.println(x: "Exit without save");  
    Logger.log(LoggerType.Supplier, logdata: "Exit without save");  
    this.mainApp.showMainView();  
}
```

```
@FXML  
void btnClickSaveExit(ActionEvent event) {  
    SerializationService.serializeEnterpriseData(manufacturers, file: "manufactures.ser");  
    SerializationService.serializeEnterpriseData(customers, file: "customers.ser");  
    SerializationService.serializeProcessCellData(processCells);  
    System.out.println(x: "Exit");  
    Logger.log(LoggerType.Supplier, logdata: "Save and Exit");  
    this.mainApp.showMainView();  
}
```

Zudem haben wir gemäss Aufgabenstellung bei wichtigen Exceptions die Fehler geloggt. Aber das ist nicht so ne coole Story :D

Learnings

Dieses Kapitel fasst diverse Learnings aus dem Projekt zusammen. Wir haben Fehler gemacht und haben auch iterativ immer wieder an einzelnen Dingen gearbeitet. Diese Entwicklung ist hier dargestellt.

User Experience

Das selbst erstellte Spongebob-Meme von vorhin war nur ein Learning in Bezug auf UX.

Wir haben nach Nutzertests gemerkt, dass es doof ist, wenn das erste Element in einer Liste nicht ausgewählt ist. Das führt zu einem unnötigen Klick und der Nutzer findet es unintuitiv.

```
123 130      tableAreas.getSelectionModel().selectedItemProperty()
124 131          .addListener((observable, oldValue, newValue) -> showAreaInfo(newValue));
132 +      tableAreas.getSelectionModel().selectFirst();
125 133
```

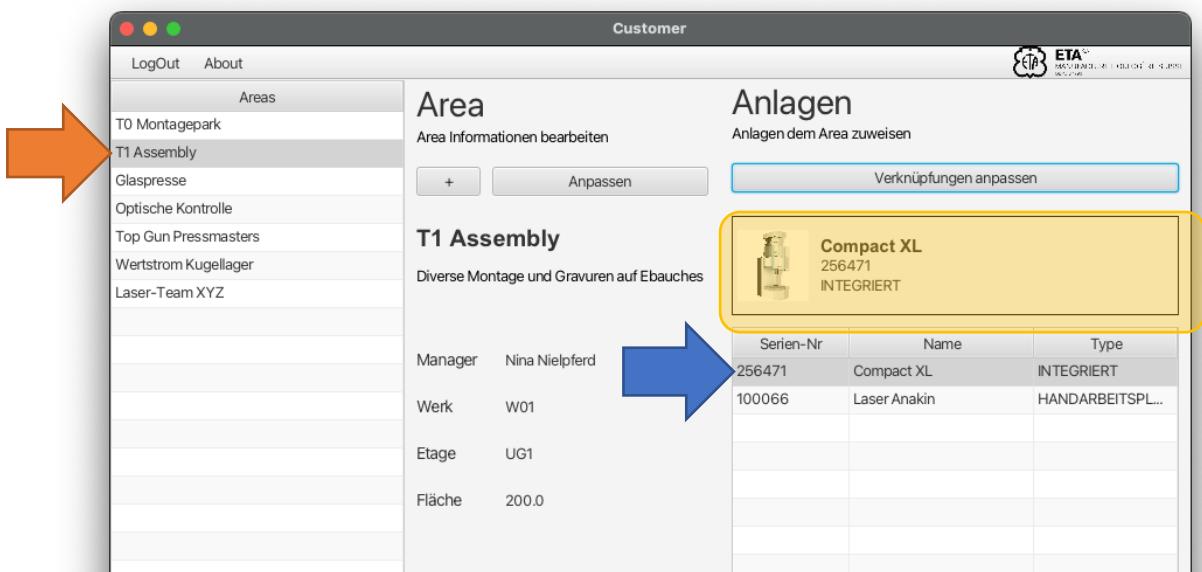


Abbildung 15 Customer View nach UX Überarbeitungen

Viele User können sich unter Zahlen nicht viel vorstellen. Daher erfassen wir ja bereits Bilder für die Anlagen. Nun aber wollten wir diese auch bei Areas darstellen. Daher haben wir nachträglich eine kleine Vorschau integriert, welche das Bild und die Attribute listet:

```
134      tableProcessCells.getSelectionModel().selectedItemProperty().addListener((observable, oldValue, newValue) -> {
135          if (newValue != null) {
136              imageProcessCell.setImage(new Image(newValue.getImagePath()));
137          +         labelProcessCellName.setText(newValue.getName());
138          +         labelProcessCellType.setText(String.valueOf(newValue.getType()));
139          +         labelProcessCellNr.setText(String.valueOf(newValue.getSerialnumber()));
140          +         System.out.println(newValue.getArea());
141      } else {
142          -         imageProcessCell.setImage(null);
143          +         setDetailTexts();
144      }
}
```

Diese Vorschau dient auch gleich als Hilfsmittel um den User anzuleiten:

```
149 +     private void setDetailTexts() {
150 +         imageProcessCell.setImage(null);
151 +         labelProcessCellName.setText("Details zur Anlage");
152 +         labelProcessCellNr.setText("Selektieren Sie eine Anlage in der Tabelle");
153 +         labelProcessCellType.setText("oder weisen Sie eine dem Area zu.");
154     }
```

Entwicklung Wireframe → App

Für uns war extrem spannend zu sehen, was wir alles nicht nach unseren ersten Wireframes und Prototypen entwickelten. Hier einige Beispiele:

Verknüpfungs-Manager

Für die Zuweisung entstand eine geniale Art und Weise, welche in unserem interaktiven Protoypen überhaupt nicht intuitiv gezeichnet war:

The image shows two side-by-side windows. On the left is a wireframe of a user interface with a dropdown menu 'Maschine', a text input 'Area 1', and two radio buttons for 'Teile-Fertigung' and 'Prototypenbau'. Below these are two buttons: 'abbrechen' and 'speichern'. On the right is the actual application window titled 'Process Cell'. It has two main sections: 'Verfügbare Anlagen' (Available Machines) and 'Zugewiesene Anlagen' (Assigned Machines). The 'Verfügbare Anlagen' section lists three machines: Laser Yoda (Type: HANDARBEITSPLATZ), Laser Anakin (Type: HANDARBEITSPLATZ), and Super Hydra... (Type: INTEGRIERT). The 'Zugewiesene Anlagen' section lists two machines: Compact 500 (Type: INTEGRIERT) and Compact XL (Type: INTEGRIERT). Between the two sections are two buttons: '<-->' and 'save'.

Customer View

Die Kundenansicht hat sich gleich mehrfach verändert. Am Ende hat sie sich immer mehr der Herstelleransicht angeglichen, um Konsistenz und Wiedererkennung im Tool zu schaffen. Diese Unklarheit ist leider erst beim Schreiben der Anleitung aufgefallen.

The image displays three versions of a customer-facing application window. The leftmost version shows a sidebar with 'CompactM Press' selected, displaying 'Unterstell Bauform' and '4 Installationen'. The middle version shows a sidebar with 'Glaspress' selected, displaying 'T0 Montagepark' and 'ProcessCells'. The rightmost version shows a sidebar with 'Glaspress' selected, displaying 'Area' information (Manager: Leo Löwe, Werk: W02, Etage: OG3, Fläche: 250.0) and a 'Anlagen' section with a table of assigned machines. The table includes rows for 'Laser Anakin' (100066, Type: HANDARBEITSPLATZ) and 'Super Hydraulic Press' (444256, Type: INTEGRIERT).

Speichern von Inhalten

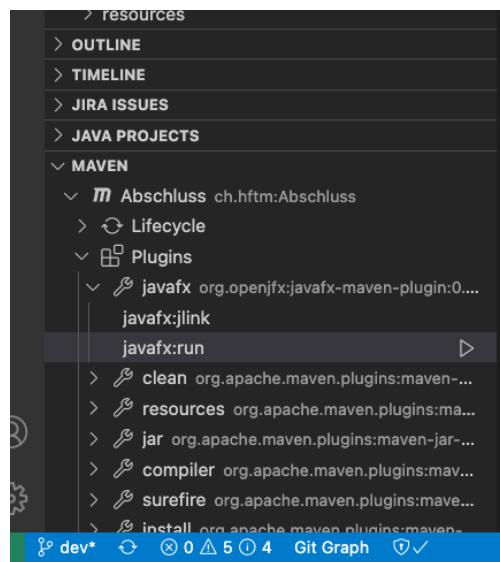
Da wir sowohl macOS als auch Windows als Operationssysteme einsetzen musste eine einheitliche Lösung für die Speicherung der Files her. Unser FolderPathService prüft dabei das verwendete OS und übergibt dementsprechend dann den String zum Speicherort.

```
public class FolderPathService {  
    public static String getProgramFolderPath() {  
        String folderName = "Maschinenverwaltung";  
        String path = "";  
        String os = System.getProperty(key: "os.name").toLowerCase();  
        if (os.contains(s: "win")) {  
            path = System.getenv(name: "APPDATA") + "\\\" + folderName;  
        } else if (os.contains(s: "mac")) {  
            path = System.getProperty(key: "user.home") + "/Library/Application Support/" + folderName;  
        } else {  
            System.err.println(x: "Unsupported operating system.");  
            System.exit(status: 1);  
        }  
        File folder = new File(path);  
        if (!folder.exists()) {  
            folder.mkdir();  
        }  
        return path;  
    }  
}
```

Starten der Applikation

Im Entwicklungsmodus muss die Applikation via Maven>Plugins>javafx>javafx:run gestartet werden, da wir es nicht fertig gebracht haben den normalen debug-runner in vs code zum Laufen zu bringen... :/

Mit einer älteren Java FX Version in den Maven Dependencies klappt es, aber wir haben uns entschieden die neuste Version zu nutzen.



Benutzerhandbuch

Dieses Kapitel dokumentiert die Nutzung der «Maschinenverwaltung» und kann als User-Dokumentation gesehen werden. Sie ist auch als Showcase der Features geeignet.

Das System ist für zwei primäre Stakeholder entwickelt worden. Wir unterscheiden:

- Kunde (Customer)
- Hersteller (Supplier)

Dabei sind beide Ansichten gleichermaßen aufgebaut:

The screenshot displays the 'Customer' view of the application. At the top, there's a navigation bar with 'Logout' and 'About' options. The main title is 'Menubar mit Logo des Enterprise'. On the right side of the title, there's a logo for 'ETA' and some smaller text. The left side features a sidebar titled 'Areas' with a list of locations: T0 Montagepark, Glasspresse, T1 Assembly, Wertstrom Kugellager, Glaspresso, Top Gun Pressmasters, Optische Kontrolle, and Laser-Team XYZ. Below this, a green box contains the text 'Liste der Objekte'. The central area has two main sections: 'Area' and 'Anlagen'. The 'Area' section shows details for 'T0 Montagepark', including its manager (Sascha Seepferd), work location (W01), floor (EG), and area size (16.0). The 'Anlagen' section shows a list of machines: Compact 500 (124335, INTEGRIERT), Compact XL (256471, INTEGRIERT), and Laser Anakin (100066, HANDARBEITSPL...). A callout box highlights the machine 'Compact 500' with the text 'Details des selektierten Objektes in zwei Spalten aufgeteilt.' (Details of the selected object are divided into two columns).

Kunden-Ansicht

Sie sind in der Industrie tätig und produzieren Produkte mit diversen Maschinen an unterschiedlichen Standorten. Diese Maschinen beziehen Sie von einem Hersteller, welcher auch Ihr Partner für die Wartung ist.

Damit die Unterhalts-Mechaniker des Herstellers die betroffenen Maschinen in Ihren Gebäuden wieder finden, müssen Informationen ausgetauscht werden. Dieser Austausch ist mit der "Maschinenverwaltung" jetzt ganz einfach.

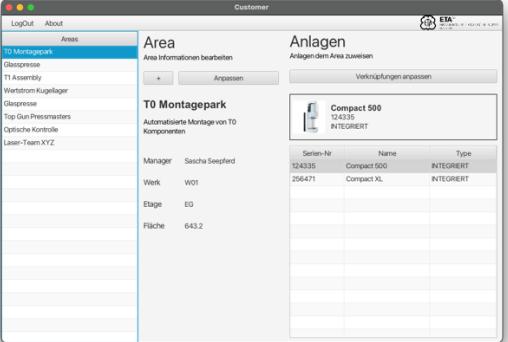
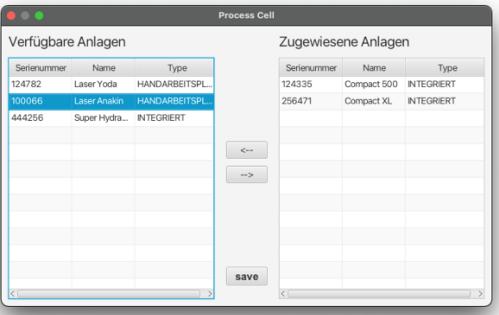
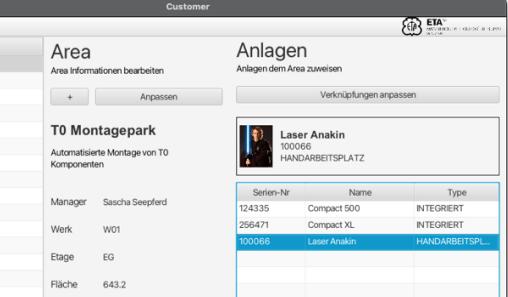
Einloggen als Kunde

Sie möchten sich als Kunde in der «Maschinenverwaltung» einloggen.

	<ol style="list-style-type: none">1. Starten Sie die Applikation.2. Tippen Sie in der Demo der Maschinenverwaltung auf die Schaltfläche «Kunde».
---	---

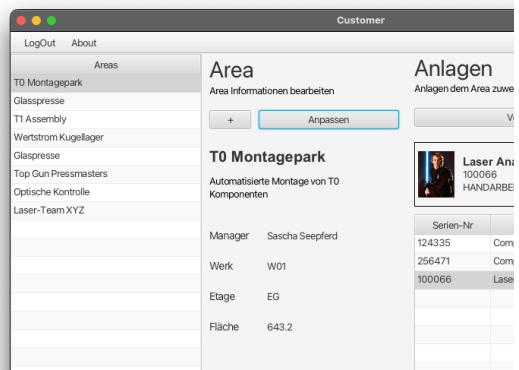
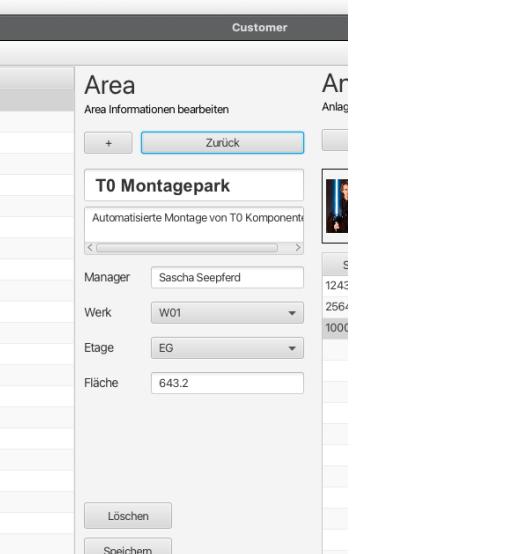
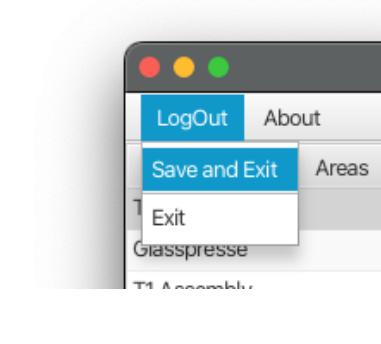
Anlage einem Bereich zuweisen

Sie sind als Kunde eingeloggt und möchten eine neue Anlage einem bereits erstellten Bereich zuweisen.

	<ol style="list-style-type: none">1. Selektieren Sie das Area, welchem Sie eine bereits erfasste Anlage Ihres Partners (Hersteller) zuweisen möchten.2. Öffnen Sie den Verknüpfungs-Manager durch klicken auf die Schaltfläche «Verknüpfungen anpassen».
	<ol style="list-style-type: none">3. Wählen Sie in der Liste «Verfügbare Anlagen» die gewünschte Anlage anhand der Seriennummer und den Hilfs-Attributen Name und Type aus.4. Klicken Sie auf den Pfeil nach Rechts um eine Anlage dem Area zuzuweisen. Mit dem Pfeil nach Links können Sie die Anlage dem Area wieder entfernen.5. Klicken Sie auf «save» um die Aktion abzuschliessen und die Änderungen zu speichern.
	<ol style="list-style-type: none">6. Die zugewiesenen Anlagen können Sie in der Liste im Bereich «Anlagen» verifizieren und erhalten jeweils eine Vorschau mit Bild, Name, Seriennummer und Type der Anlage.

Area Informationen bearbeiten

Sie möchten Area Informationen eines bereits erfassten Arbeitsbereiches in der Maschinenverwaltung anpassen.

	<ol style="list-style-type: none"> Wählen Sie in der Liste Areas die anzupassende Anlage. Klicken Sie auf die Schaltfläche «Anpassen», um die Attribute dieses Areas zu bearbeiten.
	<ol style="list-style-type: none"> Editieren Sie die Textfelder oder wählen Sie andere Optionen aus den Auswahlboxen aus. <p>Hinweis: Beachten Sie, dass die einzelnen Felder bestimmte Vorgaben verlangen. Das Formular kann nicht gespeichert werden, solange diese inkorrekt oder leer ausgefüllt sind.</p> <ol style="list-style-type: none"> Speichern Sie die Modifikationen durch klicken auf die Schaltfläche «Speichern».
	<p>Falls Sie Daten falsch erfasst haben, weist Sie das System auf die fehlerhaften Inputs hin. Korrigieren Sie diese Daten und versuchen Sie erneut zu speichern.</p>
	<p>WICHTIG:</p> <p>Damit die Änderungen persistent gespeichert werden, muss das Programm mit «Save and Exit» beendet werden.</p> <p>Mit «Exit» gehen werden keine Daten übernommen und das Programm wird geschlossen ohne zu speichern.</p>

Hersteller-Ansicht

Als Hersteller möchten Sie Ihre verkauften Anlagen mit der Seriennummer im System erfassen und diese den Kunden zuordnen. Dadurch stehen die Maschinen den Kunden in der «Maschinenverwaltung» zur Zuweisung an einem Area zur Verfügung.

Sie schaffen so eine transparente Kommunikation mit dem Kunden und schaffen Mehrwert durch den Zuschnitt des besten Unterhalts-Service. Ihr Angebot können Sie dank Statistiken und Informationen zur Verwendung Ihrer Maschinen optimieren.

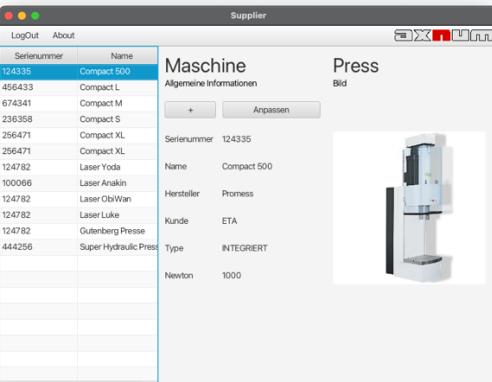
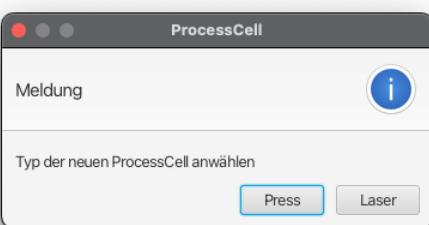
Einloggen als Hersteller

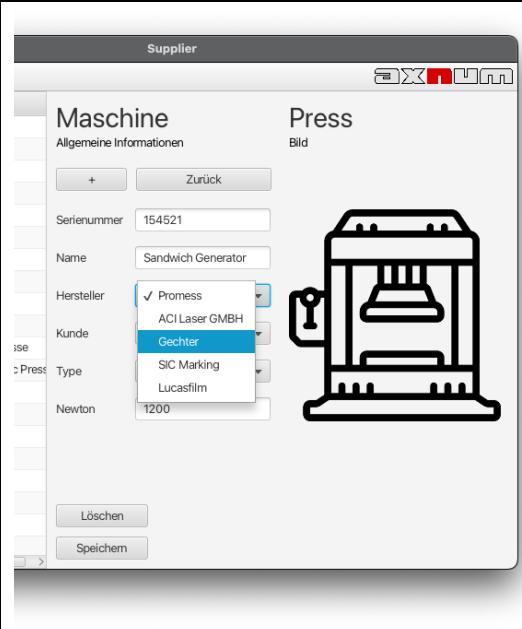
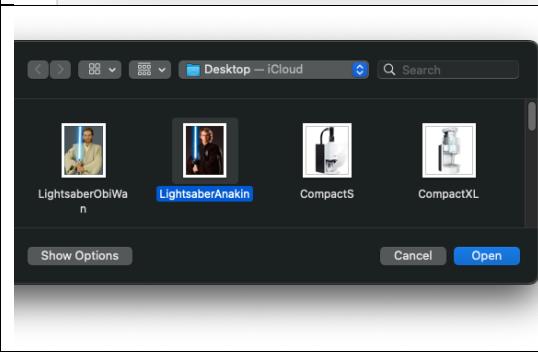
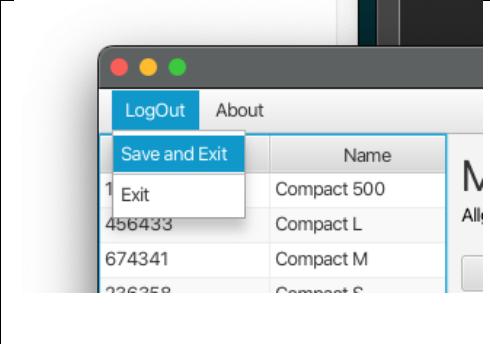
Sie möchten sich als Hersteller in der «Maschinenverwaltung» einloggen.

	<ol style="list-style-type: none">1. Starten Sie die Applikation.2. Tippen Sie in der Demo der Maschinenverwaltung auf die Schaltfläche «Hersteller».
---	--

Neue Anlage erfassen

Sie möchten sich als Hersteller in der «Maschinenverwaltung» einloggen.

	<ol style="list-style-type: none">1. In der Liste sehen Sie alle bereits erfassten Anlagen.2. Zur Erfassung einer neuen Anlage, tippen Sie auf das «+» unterhalb des Bereiches «Maschine».
	<ol style="list-style-type: none">3. Wählen Sie nun die Anlagen-Art aus. Sie können in dieser Version der Applikation zwischen Press und Laser wählen.4. Ein leeres Formular öffnet sich, welches mit einem Standard-Bild vor-ausgefüllt wird.

	<p>5. Füllen Sie alle weiteren Felder aus.</p> <p>Hinweis: Beachten Sie, dass die einzelnen Felder bestimmte Vorgaben verlangen. Das Formular kann nicht gespeichert werden, solange diese inkorrekt oder leer ausgefüllt sind.</p> <p>Speichern Sie die Modifikationen durch klicken auf die Schaltfläche «Speichern».</p>
	<p>6. Falls Sie Daten falsch erfasst haben, weist Sie das System auf die fehlerhaften Inputs hin. Korrigieren Sie diese Daten und versuchen Sie erneut zu speichern.</p>
	<p>7. Sie können ein optionales Vorschau-Bild einfügen. Klicken Sie dafür noch im Bearbeitungs-Modus auf die Schaltfläche «Bild auswählen».</p> <p>8. Wählen Sie ein Bild aus Ihrem Verzeichnis aus und klicken Sie auf «Open».</p>
	<p>WICHTIG:</p> <p>Damit die Änderungen persistent gespeichert werden, muss das Programm mit «Save and Exit» beendet werden.</p> <p>Mit «Exit» gehen werden keine Daten übernommen und das Programm wird geschlossen ohne zu speichern.</p>

Ihre Daten stehen so den Kunden zur Zuweisung an deren Areas zur Verfügung, wo dann ganz tolle Dinge mit den Anlagen gemacht werden können :D

Supplier

LogOut About

Seriennummer Name

124335	Compact 500
456433	Compact L
674341	Compact M
236358	Compact S
256471	Compact XL
256471	Compact XL
124782	Laser Yoda
100066	Laser Anakin
124782	Laser ObiWan
124782	Laser Luke
124782	Gutenberg Presse
444256	Super Hydraulic Press
154521	Sandwich Generator

Maschine

Allgemeine Informationen

+ Anpassen

Press

Bild

Seriennummer 444256

Name Super Hydraulic Press

Hersteller SIC Marking

Kunde Ypsomed

Type INTEGRIERT

Newton 15000



Abbildung 1 Problemlösungszyklus.....	8
Abbildung 2 Requirements Engineering System, Kontext und irrelevante Umgebung	10
Abbildung 3 Produkte zum Beschriften von AxNum	5
Abbildung 4 Industrienorm ISA-88, Maschinenaufbau	11
Abbildung 5 ISA-95 Daten-Pyramide mit welchen die «Maschinenverwaltung» zusammen agiert.	7
Abbildung 6 Use Case Diagramm	12
Abbildung 7: Wireframe Entwurf erstellt mit Penpot, vor Realisierung der App.....	14
Abbildung 8: Klassendiagramm erstellt in Enterprise Architect im Stand vor der Realisierung der Applikation	15
Abbildung 9: Datenbankschema erstellt in VS Code mit VUERD	16
Abbildung 10 Auswahl Press / Laser zur Erstellung eines Objektes basierend auf einer abstrakten Klasse.....	20
Abbildung 11 So hat übrigens das UI vor den Optimierungen ausgesehen.	21
Abbildung 12 Feld zur Zuweisung von Laser und Pressen (Process Cells)	22
Abbildung 13 Ungültige Fehler, welche dank einem Thread rot erscheinen	24
Abbildung 14 Logfile-Story in Meme-Format	25
Abbildung 15 Customer View nach UX Überarbeitungen	26