



Java Cloud Native Abschlussprüfung  
- Jahr 2 –   
Projektdokumentation  
Gruppe 10 ProductionPlanner

|  |  |
| --- | --- |
| Autoren / Ersteller: | Christopher Hoppe, Nicolas Weltzel |
| Zeitraum der Projektarbeit: | 25.01.2023 – 03.02.2023 |
| Projektbezeichnung | Software zur Administration sowie Simulation von Produktionsstraßen für die Volkswagen AG |

Inhaltsverzeichnis

[1. Projektdefinition 3](#_Toc126320157)

[2. Projektplanung 3](#_Toc126320158)

[2.1 Zeitplanung 4](#_Toc126320159)

[3. Durchführung 4](#_Toc126320160)

[3.2 Design 5](#_Toc126320161)

[3.3 Datenmodell 7](#_Toc126320162)

[3.4 Businesslogik 9](#_Toc126320163)

[3.5 Aufbau des Frontend 10](#_Toc126320164)

[3.6 Implementierung 10](#_Toc126320165)

[3.7 Test / Qualitätssicherung 12](#_Toc126320166)

[4. Projektabschluss 13](#_Toc126320167)

[4.1 Soll / IST- Vergleich 13](#_Toc126320168)

[4.2 Fazit 13](#_Toc126320169)

[4.3 Ausblick 14](#_Toc126320170)

[5. Quellenverzeichnis 15](#_Toc126320171)

[5.1 Abbildungsverzeichnis 15](#_Toc126320172)

[5.2 Tabellenverzeichnis 15](#_Toc126320173)

[5.3 Glossar 15](#_Toc126320174)

[6. Anhang 16](#_Toc126320175)

[6.1 Userstories und Anforderungen 16](#_Toc126320176)

[Userstories 16](#_Toc126320177)

[Weitere Anforderungen (Stichpunkte): 17](#_Toc126320178)

# Projektdefinition

Ziel dieser Projektarbeit ist die Ertellung einer Software zur Verwaltung und Simulation von Produktionsstraßen in Form eines JavaFX- oder React- Clients, der dem Kunden die Möglichkeit gibt die Fertigstellung eines beliebigen Autos zu simulieren. Die Software soll in einer wartbaren und verständlichen Architektur aufgebaut werden, die sich, sofern möglich, am MVC-Modell und einer Client-Server-Architektur mit Spring Boot im Backend orientiert. Die Daten zu den Produktionsstraßen sowie den Akteuren (Roboter, Stationen und Mitarbeiter) sollen in einer passenden Datenbank in Azure abgelegt werden. Die Simulation soll im Backend stattfinden und, sobald sie gestartet ist, unabhängig vom Frontend Fahrzeuge produzieren.

## 

# 2. Projektplanung

Zur besseren Planung des Projekts wurden die Anforderungen zunächst in einzelne User Stories aufgespalten um die Umsetzung der verschiedenen Teilaspekte zu vereinfachen.

Zu diesen Stories wurden Akzeptanzkriterien festgelegt, um die Umsetzung der jeweilige Story beurteilen zu können.

Auf dieser Basis wurde ein erstes Datenmodell abgeleitet, das die Grundlage für die Gestaltung des Backends war.

Aufgrund der vielfältigen Gestaltungsmöglichkeiten, dem bewährten Zustands- und Ereignismanagment sowie der umfangreiche Erfahrung aus vergangen Projekten wurde sich für die Verwendung von React für die Umsetzun des Frontends entschieden.

Anhand dieses groben Projektentwurfs konnten wir unsere Zeitplanung skizzieren, die wir in Sprints zu je einem Tag aufgeteilt haben. Es folgt eine Übersicht über diese Sprints:

## 2.1 Zeitplanung

|  |  |
| --- | --- |
| Datum | Aufgaben |
| 25.01.2023 | * Entscheiden bezüglich der zu verwendenden Frameworks * Erstellen der User Stories aus dem Projektantrag * Erstellen der Zeit- und Sprintplanung * Mockups für die GUI erstellen * Einrichtung des Repositories und Erstellung der Spring / React Projekte |
| 26.01.2023 | * Anlegen der Dokumentation * Erstellung des Datenmodells * Start der Implementierung im Backend inkl. Tests * Erstellen Landing Page |
| 27.01.2023 | * Erstellung Ressourcen Seite * Erstellung der Endpunkte und Logik für die Ressourcen * Verbinden von Frontend und Backend * Dokumentation |
| 30.01.2023 | * Erstellen einer Seite zum Anlegen einer Produktionslinie * Erstellen der Endpunkte und Logik zur Erstellung einer Produktionsline * Dokumentation |
| 31.01.2023 | * Erstellen der Simulationslogik sowie der Endpunkte zur Steuerung * Einbindung der Simulationsdaten und Steuerung auf der Landingpage * Dokumentation |
| 01.02.2023 | * Erstellen einer Azure DB * Verbindung zwischen Azure und Backend herstellen * Backend deployen * Validierung und Bugfixing im Frontend * Dokumentation |
| 02.02.2023 | * Frontend deployen * Bugfixing / Codeconventions * Dokumentation |
| 03.02.2023 | * Schönheitskorrekturen * Bugfixing * Dokumentation |

Tabelle 1: Grobe Zeitplanung

# 3. Durchführung

Nach der groben Projektplanung wurde ein React-Client (Frontend) mit ‚npx create-react-app‘, sowie eine Java Spring Boot Application (Backend) mit dem Spring Initializr erstellt. Diese wurden dann dem vorab geklonten Gruppenrepository hinzugefügt, damit alle Projektbeteiligten Zugriff auf den gesamten Quellcode haben.

Tägliche Standups am Morgen dienten dazu die Tagesziele zu definieren. An Tagesende wurde eine Retrospektive durchgeführt, um über die Fortschritte des Tages und aufgetretene Probleme zu sprechen und diese, falls möglich, in den nächsten Sprints abzustellen.

## 3.2 Design

Das Design der App sollte möglichs einfach und intuitiv sein. Daher entschieden wir uns dafür, möglichst viele der relevanten Informationen bereits auf der Startseite anzuzeigen. Um den User auch in den anderen Bereichen der Anwendung sofort abzuholen, wurde über alle Seiten ein einheitlicher Header verwendet, über den auch die Navigation abläuft. Insgesamt verfügt die Anwendung über drei Ansicheten, die im folgenden kurz vorgestellt werden:

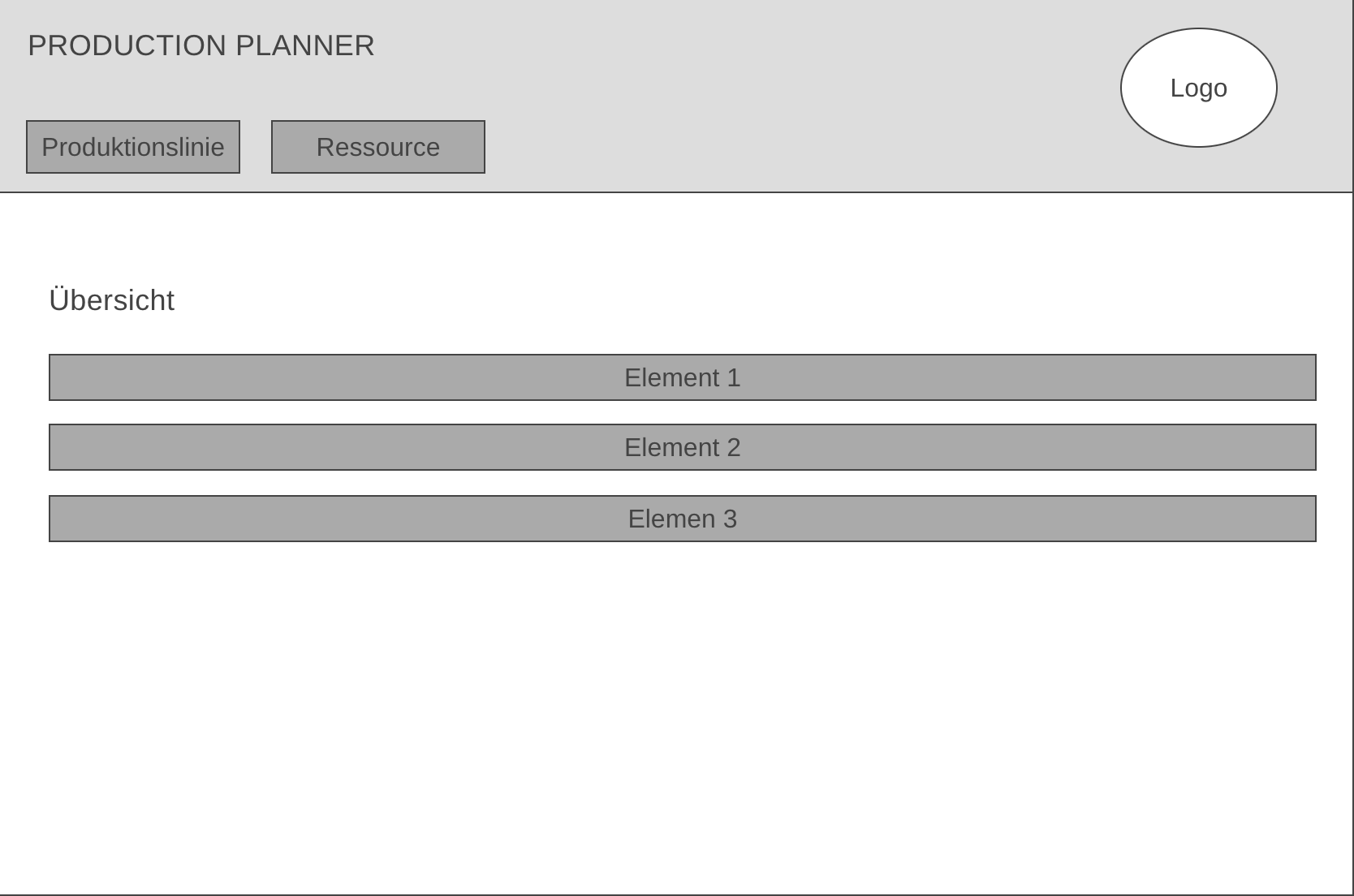


Abbildung 1: Mockup Startseite

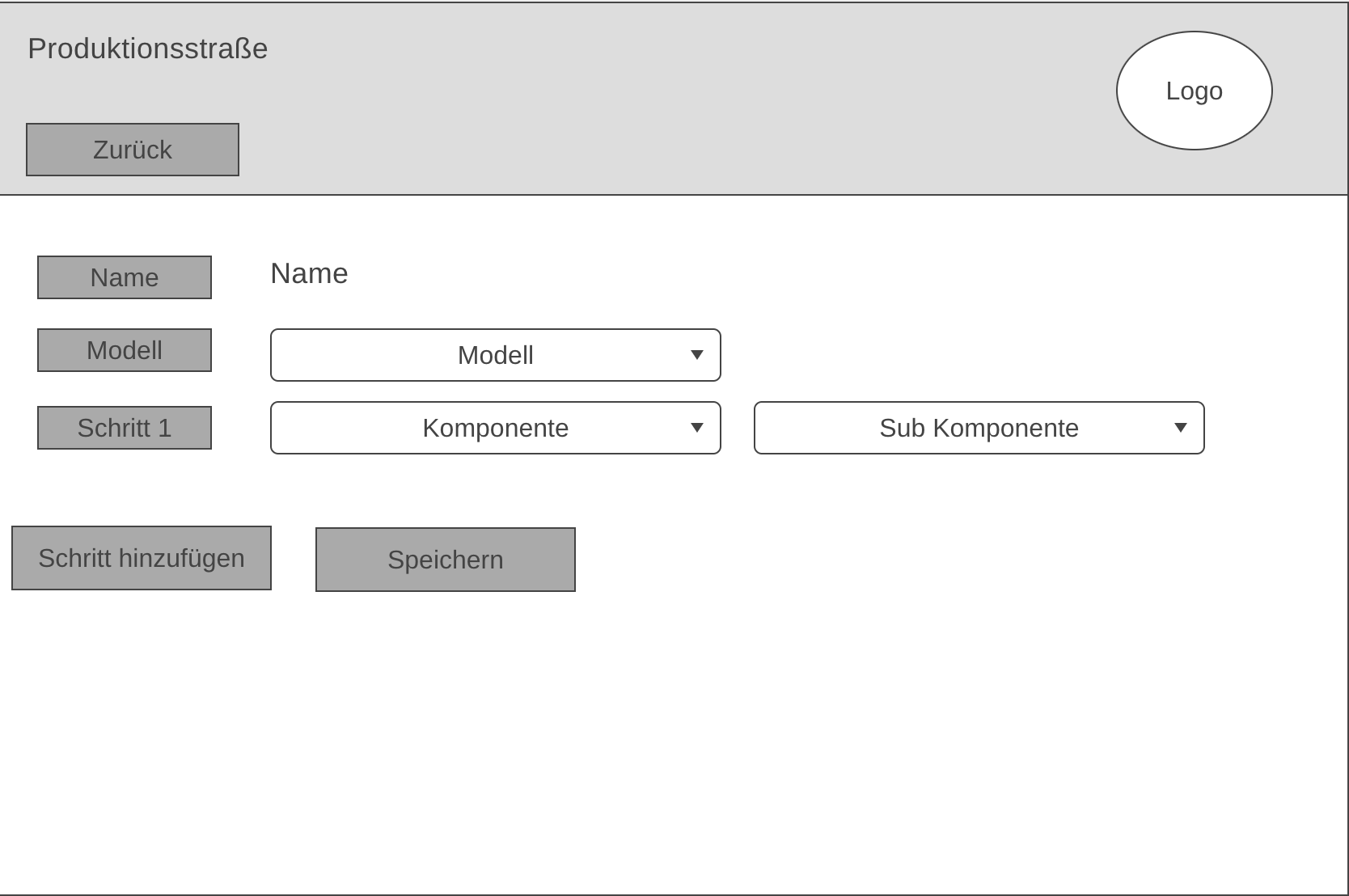
Wie in Abbildung 1 dargestellt soll diese Ansicht den Mittelpunkt der Anwendung darstellen. Sie soll einen Überblick über die Erstellten Produktionslinien bieten und gegebenenfalls und weitere Elemente zur Darstellung des aktuellen Status (läuft / läuft nicht, Produzierte Fahrzeuge etc.) und zur Steuerung der Simulation (Start, Stopp, Geschwindigkeit) enthalten. Darüber hinaus kann von hier zu den Seiten zur Erstelleung der Ressourcen und zum Anlegen einer neuen Produktionsline navigiert werden. 

Abbildung 2: Mockup Produktionsstraße erstellen

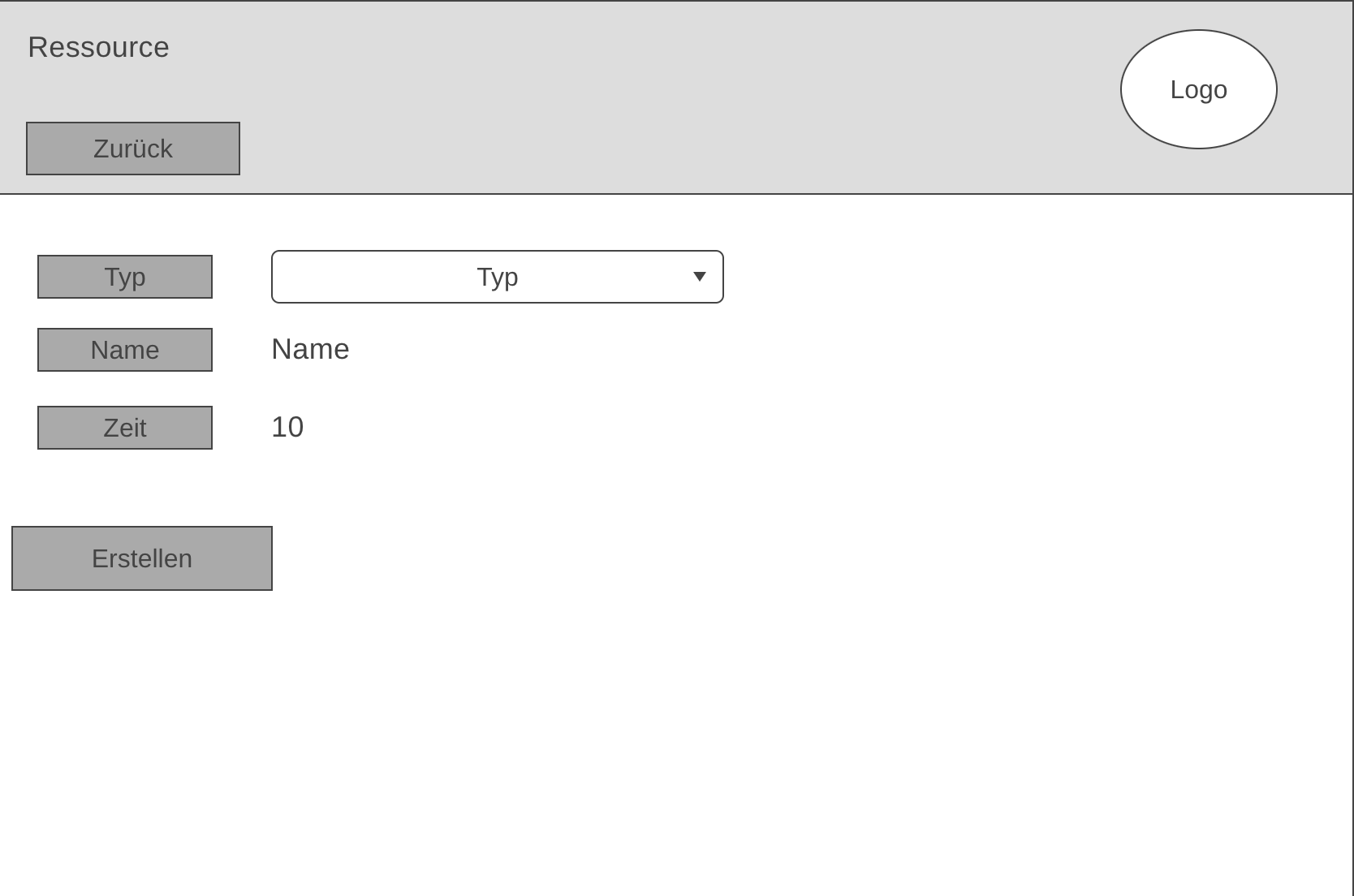
Abbildung 2 und Abbildung 3 sollen die jeweiligen Unterseiten zur Erstellung einer Produktionslinie bzw. der zuzuordnenden Ressourcen (Roboter, Stationen und Menschen) darstellen. Hier kann der Nutzer, je nach Attribut, aus Auswahlfeldern oder durch Eingabe eines Freitexts neue Elemente erstellen.

Abbildung 3: Mockup Ressource erstellen

Über alle Eingabefelder soll es eine Validierung geben, um sicherzustellen, dass nur zulässige Werte abgesendet werden können. Darüber hinaus soll der User Feedback zum Status seiner Eingaben bzw. Anfragen erhalten. Bei der Nutzung ist zu beachten, dass eine Navigation ausschließlich über die zur Verfügung gestellten Elemente erfolgen muss. Beim direkten Aufruf einer Unterseite, stehen ggf. nicht alle Daten bereit und es kann zu Einschränkungen kommen.

## 3.3 Datenmodell

Das Datenmodell dreht sich um die zentrale Komponente des Programms, der Produktionslinie (siehe Tabelle 2). Innerhalb dieser Klasse wird bestimmt und festgelegt ob diese simuliert werden kann, und falls ja wie die Parameter dafür ausgeprägt sind. Darüber hinaus sind hier alle weiteren Unterkomponenten wie z.B. die Stationen und Roboter hinterlegt.

|  |  |  |
| --- | --- | --- |
| Name | Datentyp | Zweck |
| id | long | Eindeutige Identifikation |
| name | String | Name der Komponente |
| isRunnable | boolean | Kennzeichnung der Lauffähigkeit |
| isActive | boolean | Kennzeichnung, ob eine Simulation mit dieser Komponente aktiv ist |
| simSpeed | int | Simulationsgeschwindgkeit |
| simTime | float | Simulationszeit |
| timeToCompletion | long | Benötigte Simulationszeit zur Fertigstellung einer Komponente |
| finishedParts | long | Fertiggestellte Komponenten in dieser Simulation |
| carModel | CarModel | Zuordnung eines Fahrzeugmodells |
| components | List<ProductionLineComponent> | Zuordnung der Unterkomponenten |

Tabelle 2: Klasse ProductionLine

|  |  |  |
| --- | --- | --- |
| Name | Datentyp | Zweck |
| id | long | Eindeutige Identifikation |
| name | String | Name der Komponente |
| step | long | Schritt im Produktionsablauf |
| productionTime | long | Dauer des Schritts |
| isOnDuty | Boolean | Komponente ist in einer Linie im Einsatz |
| employees | List<Employee> | Zugeordnete Arbeiter |
| type | Type | Art der Komponente |
| productionLine | ProductionLine | Zugeordnete Produktionslinie |

Tabelle 3: Klasse ProductionLineComponent

Die Attribute der einzelnen Bestandteile einer Produktionslinie sind in Tabelle 3 dargestellt.Hier wird festgelegt wie lange es dauert diesen Produktionsschritt durchzuführen, von welchem Typ die Komponente ist (Roboter oder Station) und ob sie bereits im Einsatz ist (inklusive Zuweisung zu einer Produktionslinie).

Neben der *ProductionLineComponent* wird einer Produktionslinie noch ein Fahrzeugmodell (siehe Tabelle 4) zugewiesen. In diesem werden die einzelnen Fahrzeuge definiert, die über Ihre Komplexität die Dauer eines Produktionszyklus beeinflussen.

|  |  |  |
| --- | --- | --- |
| Name | Datentyp | Zweck |
| id | long | Eindeutige Identifikation |
| name | String | Name der Komponente |
| complexity | float | Komplexitätsgrad der Komponente (beeinflusst die Zeit zur Fertigstellung) |

Tabelle 4: Klasse CarModel

Eine *ProductionLineComponent* vom Typ Station kann nur arbeiten wenn ihr mindestens ein Mitarbeiter (Tabelle 5) zugeordnet wird:

|  |  |  |
| --- | --- | --- |
| Name | Datentyp | Zweck |
| id | long | Eindeutige Identifikation |
| name | String | Name der Komponente |
| isOnDuty | boolean | Komponente ist in einer Station im Einsatz |

Tabelle 5: Klasse Employee

Die Zusammenhänge der einzelnen Klassen lassen sich am besten aus nachfolgendem ER-Diagramm (Abbildung 4 - ER-Diagramm der Datenstruktur) erkennen:

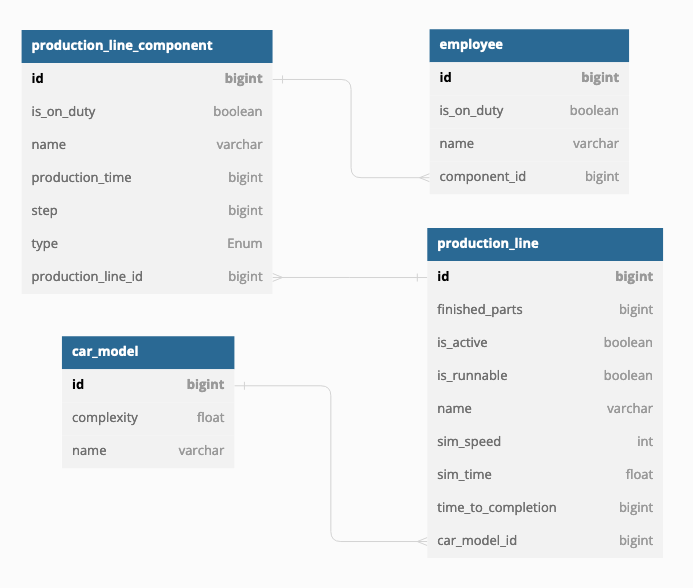


Abbildung 4: ER-Diagramm der Datenstruktur

## 3.4 Businesslogik

Beim Anlegen einer Produktionsstraße wird diese validiert. Dabei wird überprüft ob die Vorraussetzung für das Starten einer Simulation gemäß der Anforderungen erfüllt sind. Es werden dafür die folgenden Punkte geprüft:

* Sind mindestens Drei Komponenten vorhanden?
* Ist die zu simulierende Geschwindigkeit größer Null?
* Ist die Komplexität des hinterlegten Fahrzeugs im Bereich von [0.5,1.5]?
* Ist jeder Komponente vom Typ *station* mindestens ein Mitarbeiter zugeordnet?

Sind diese Bedingungen erfüllt wird das Attribut isRunnable auf true gesetzt und die *productionTime* der Komponenten addiert und als *timeToCompletion* gespeichert.

Beim erfolgreichen Starten einer Simulation wird die Produktionslinie, im SimulationService in die Liste der aktiven Simulationen aufgenommen. Für alle Elemente dieser Liste wird über einen *@Scheduled* Annotation im Zyklus von einer Sekunde die Sumulation ausgeführt.

Dabei wird die Methode *addSimTime* der Produktionslinien aufgerufen. Übergeben wird die Grundgeschwindigkeit der Simulation, die über den Parameter BASE\_SIM\_TIME, parametriert werden kann. In dieser Methode wird zunächst die Simulationszeit um den Übergabewert erhöht. Dieser wird noch mit der internen Simulationsgeschwindigkeit der Produktionslinie *simSpeed* multipliziert, und durch die Komplexität des Fahrzeugmodells dividiert. Anschließend wird geprüft ob die gesammelte Simulationszeit *simTime* ausreichend ist, um ein Fahrzeug fertig zu stellen. Ist dies der Fall wird die Fertigstellungszeit *timeToCompletion* von *simTime* abgezogen (bei Bedarf auch mehrfach) und die Anzahl der Fertigen Fahrzeuge *finishedParts* entsprechend erhöht.

Für die gewählte Datenstruktur bietet sich die Umsetzung in einer relationalen Datenbank an. Aufgrund der einfachen integration in Azure und den Erfahrungen der Entwickler in vergangenen Projekten wurde für eine mySQL Datenbank verwendet.

Mit Spring Boot und Spring Data konnte das Datenmodell anhand der Annotationen im Quellcode automatisch auch in der Datenbank mit den entsprechenden Verbindungen der Entitäten zueinander erstellt werden. So konnten möglich Fehlerquellen bei der händischen definition von SQL-Statements ausgeschlossen werden, und der Fokus auf die weitere Logik gelegt werden.

## 3.5 Aufbau des Frontend

Auf der Startseite (Abbildung 5) bekommt der User eine Übersicht über alle aktiven Produktionslinien. Für Linien die nicht aktiv sind (Simulationsstatus: läuft nicht) kann er das Fahrzeugmodell über das Dropdown in der Spalte Produziert ändern und die Geschwindigkeit über die Spalte Speed.

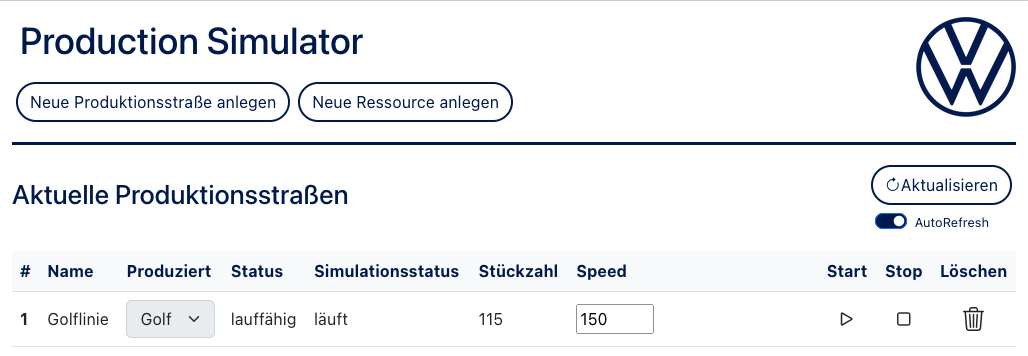


Abbildung 5: Startseite

Der Status gibt an ob eine Simulation gestartet werden kann (dies wird bereits beim Erstellen geprüft). Sind alle Vorraussezungen erfüllt, kann der User über die Steuerungselement die Simulation Starten bzw. Stoppen. Ein Löschen ist nur möglich, wenn die Simulation grade nicht läuft. Beim Löschen ist zu beachten, dass alle Ressourcen der Station mit gelöscht werden.

In Abbildung 6 ist der Dialog zum Anlegen einer Produktionsstraße dargestellt. Hier können über die Entsprechenden Textfelde und Dropdowns Name, Modell und Komponenten ausgewählt werden. Wird eine Station ausgewählt erscheint zusätzlich ein Dropdown, über den verfügbare Mitarbeiter ausgewählt, und über den Button rechts davon der Station zugeordnet werden können. Die Zugeordneten Mitarbeit werden unmittelbar unter der Station aufgelistet.

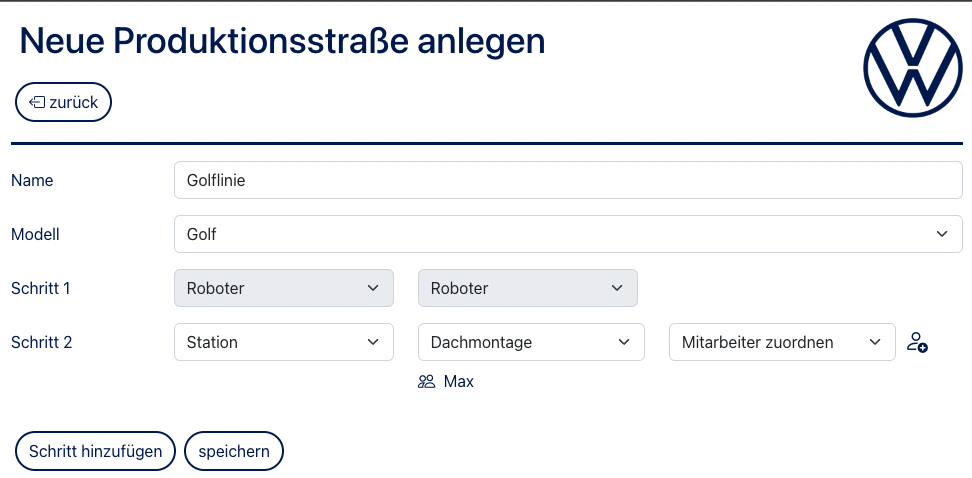


Abbildung 6: Produktionsstraße anlegen

## 3.6 Implementierung

Die Umsetzung erfolgte zum Teil vor Ort im Mobile Life Campus, zum Teil remote von zu Hause. Das Vorgehen war jedoch grundsätzlich sehr ähnlich. Überwiegend wurde im Pair- programming entwickelt. Dazu wurde im Frontend Visual Studio Code als DIE verwendet, welches über das „Liveshare“ Plugin die Möglichkeit bot, zeitgleich am Code zu arbeiten und auch die Ergebnisse gemeinsam über geteilte Server zu betrachten.

Für das Backend wurde IntelliJ in der Community Edition verwendet. Diese bot zwar mit dem Plugin „Code with me“ ähnliche Funktionalitäten war jedoch aufgrund der Lizenz nur für kurze Sessions von bis zu 30 Minuten einsetzbar, was zu deutlichen Ineffizienzen geführt hat. Aus diesem Grund sind wir für die Implementierung des Backends so weit möglich dazu übergegangen an einem Rechner zu entwickeln.

Durch die Arbeit im Pair, war eine sehr direkte Kommunikation gegeben, die entweder vor Ort, oder über die Tools Microsoft Teams bzw. Big Blue Button stattfand. Dadurch konnte jederzeit Feedback zu einzelnen Funktionen eingeholt werden, Fehler frühzeitig erkannt und behoben werden und damit die Qualität der Implementierung verbessert werden.

Während der Implementierung lag ein starker Fokus darauf, einzelne Features zu commiten, um möglichst jederzeit einen Abgeschlossenen funktionierenden Stand zu gewährleisten und einzelne Änderungen möglichst nachvervfolgbar zu machen.

Insbesondere bei der Implementierung des Backends kam es allerdings auch zu größeren Schwierigkeiten, da die Persistierung der Produktionslinien Objekte mit angehängten Kindelementen (Station, Roboter) und deren Kindelementen (Angestellte) zu vielfältigen Fehlern geführt hat. Diese Fehler konnten nicht über die Konfiguration von Spring / Hibernate behoben werden und mussten über einen Umweg bei der Speicherung gelöst werden. Dazu war es nötig, die jegliche Referenzen auf andere Objekte aus den Produktionslinien zu lösen und über die entsprechenden Repositorys zu persistieren. Danach konnte die Produktionslinie ohne Referenzen persisitiert werden, und darauf konnten die Objekte wieder zusammengesetzt werden.

Im Frontend war es zudem eine große Herausforderung beim Anlegen der Produktionslinien zunächst die Auswahl über alle Komponenten zu ermöglichen, um dann, wenn ein Schritt abgeschlossen und hinzugefügt wurde, die im nächsten Schritt zur Verfügung stehenden Elemente entsprechend zu reduzieren, ohne dabei auch die vorher getroffene Auswahl zu verändern. Abbildung 5 zeigt dieses Problem: Roboter R10XY ist in Schritt 1 ausgewählt und soll in Schritt 2 nicht mehr als Auswahl zur Verfügung stehen.

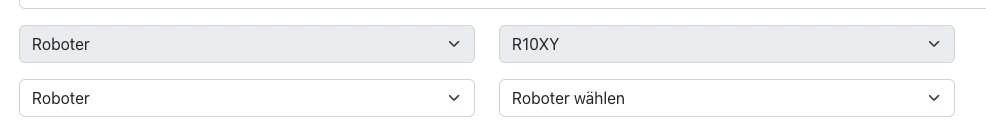


Abbildung 7: Auswahl von Komponenten

## 3.7 Test / Qualitätssicherung

Bei der Teststrategie wurde der Fokus auf die Endpunkte und die Datenverarbeitung des Backends gelegt. Mit Integrationstests konnte der Gesamte Lebenszyklus der Daten von Datenbank bis zur API und umgekehrt abgesichert werden und auf spezifischere Tests (z.B. Unit Test) verzichtet werden. Dabei wurden generische Komponenten außen vor gelassen und ausschließlich die komplexen Objekte wie die Produktionslinien und die Simulation getestet (die wiederum die simpleren Objekte beinhalten).

Um eine gleichbleibende Testumgebung sicherzustellen und nicht mit den produktiven Daten in Konflikt zu kommen, wurden die Tests in einer H2 Datenbank durchgeführt, die nach jedem Durchlauf zurückgesetzt wird.

Darüber hinaus wurde über die gesamten Entwicklung auf Fehler und Bugs geachtet. Besonders im Frontend wurde gezielt nach diesen gesucht und versucht unzulässige Zustände zu erreichen. Gefundene Fehler wurden dann nach Möglichkeit direkt im Pairprogramming gelöst.

Nach Abschluss der Implementierung, wurden alle Funktionen durch die Entwickler getestet. Dabei wurden insbesondere Einschränkungsregeln, sowie ähnliche Grenzzustände erprobt. Hauptziel des Testens war es sicherzustellen, dass jede Komponente der Software ihre Funktionalität zuverlässig erfüllt.

# 4. Projektabschluss

## 4.1 Soll / IST- Vergleich

Die grobe Zeitplanung zu Beginn des Projekts konnte nicht eingehalten werden. Die in der Implementierung beschriebenen Bugs und Herausforderungen traten in der Mitte des Projekts auf und hatten eine deutliche Verzögerung zur Folge, die dann in den folgenden Tagen aufgearbeitet werden musste. Dadurch mussten auch Abstriche bei weiteren (optionalen) Funktionalitäten geamcht werden. Die geforderten Kernfunktionen, konnten jedoch vollständig umgesetzt werden. Trotz allem blieb am Ende des Projekts genug Zeit um den Code abschließend zu strukturieren und zu optimieren.

Bei der grafischen Umsetzung gab es kaum Abweichungen zu der finalen Umsetzung, das Konzept hat sich als gut erwiesen.

## 4.2 Fazit

Trotz der kurzen Projektzeit und den dafür sehr vielfältigen und komplexen Anforderungen, insbesonder an das Datenmodell, konnten alle Anforderungen an die Software implementiert und darüber hinaus einige Komfortfeature umgesetzt werden. Insbesondere hinsichtlich Funktionalität, Benutzerfreundlichkeit und Design konnten alle Funktion zur Zufriedenheit der Entwickler umgesetzt werden und es musste nicht auf rudimentäre Gestaltung zurückgegriffen werden.

Die Zusammenarbeit lief ausgesprochen produktiv und beide Teammitglieder konnten voneinander lernen und sich gegenseitig unterstützen. Dabei konnten komplexe Konzepte implementiert werden, die für beide Partner neu waren.

Die initiale Planung der Struktur hat sich als sehr hilfreich für die Umsetzung erwiesen, auch wenn nicht alles wie gewünscht umgesetzt werden konnte.

Die Entwicklung des Frontends mit React und des Backends mit Java Spring Boot bot die Möglichkeit unser Wissen in beidem zu vertiefen und im Umgang mit diesen Technologien sicherer zu werden.

Durch die Erfüllung der Anforderungen und die persönlichen Erfahrungen, kann das Projekt als voller Erfolg betrachtet werden.

## 4.3 Ausblick

Obwohl alle Anforderungen an die Software erfüllt wurden konnten, gibt es weiteres Verbesserungspotenzial sowie weitere Funktionen die eine sehr sinnvolle Erweiterung für die Anwendung darstellen könnten.

Zunächst sollte die im Teil Implementierung beschrieben Problem bei der Persistierung der Produktionslinien gelöst werden. Dann wäre es sinnvoll die folgenden weiteren Feature zu implementieren die aufgrund der Kürze der Zeit nur vorbereitet aber nicht umgesetzt werden konnten:

* Eine Simulation kann pausiert und fortgesetzt werden.
* Eine Simulation kann nachträglich angepasst werden.
* Ressourcen können nachträglich angepasst werden.
* Wenn mehrere Mitarbeiter einer Station zugeordnet werden, sollte sich die Verarbeitungsgeschwindigkeit dieser Station verbessern.

# 5. Quellenverzeichnis

## 5.1 Abbildungsverzeichnis

[Abbildung 1: Startseite 5](#_Toc126322585)

[Abbildung 2: Produktionsstraße erstellen 6](#_Toc126322586)

[Abbildung 3: Ressource erstellen 6](#_Toc126322587)

[Abbildung 4: ER-Diagramm der Datenstruktur 9](#_Toc126322588)

[Abbildung 5: Startseite 10](#_Toc126322589)

[Abbildung 6: Produktionsstraße anlegen 11](#_Toc126322590)

[Abbildung 7: Auswahl von Komponenten 13](#_Toc126322591)

## 5.2 Tabellenverzeichnis

[Tabelle 1: Grobe Zeitplanung 4](#_Toc126322592)

[Tabelle 2: Klasse ProductionLine 7](#_Toc126322593)

[Tabelle 3 Klasse ProductionLineComponent 7](#_Toc126322594)

[Tabelle 4: Klasse CarModel 8](#_Toc126322595)

[Tabelle 5: Klasse Employee 8](#_Toc126322596)

## 5.3 Glossar

API Application Programming Interface

Bug Programmfehler

Bug Fixing Beheben von Programmfehlern

MVC Pattern Model-View-Controller Pattern.

Pairprogramming Zwei Entwickler programmieren im Vier-Augen-Prinzip.

Repository Projektarchiv in der Versionsverwaltungs-software GIT

React JavaScript-Softwarebibliothek

Retrospektive Bezeichnung aus der Scrum Vorgehensweise um die bisherige Arbeitsweise zu reflektieren und zukünftig zu optimieren

Scrum Framework für das Projektmanagement nach agilen Prinzipien

Spring Initializr Tool zum Erstellen von Spring Boot Applikationen

Standup Kurzes Zusammenfinden des Projektteams um Tagesziele zu definieren.

# 6. Anhang

## 6.1 Userstories und Anforderungen

### Userstories

#### Userstory 1:

Als Anwender möchte ich, wenn ich die Startseite aufrufe, eine Übersicht über die angelegten Produktionslinien angezeigt bekommen.

#### Akzeptanzkriterien:

* Es können beliebig viele Varianten angezeigt werden.
* Es wird eine Name, ein Simulationsstatus und die Lauffähigkeit angezeigt.

#### Userstory 2:

Als Anwender möchte ich über die Startseite die Simulationen steuern können.

Akzeptanzkriterien:

* Ich kann eine Simulation starten
* Ich kann eine Simulation stoppen
* Ich kann die Geschwindigkeit anpassen

#### Userstory 3:

Als Anwender möchte ich den Status der Simulationen aktualisieren können.

#### Akzeptanzkriterien:

* Ich kann über einen Schalter die aktuellen Status einerProduktionslinie abrufen.
* Ich kann den abruf zyklisch ablaufen lassen.

### Weitere Anforderungen (Stichpunkte):

* Seite zum Anlegen von Ressourcen
  + Roboter, Station, Mitarbeiter, Fahrzeugmodell
  + Fehler werden beim Speichern angezeigt
* Seite zum Anlegen einer Produktionsstraße
  + Ressourcen können den verschiedenen Schritten hinzugefügt werden
  + Abgeschlossene Schritte werden ausgegraut / können nicht verändert werden
  + Nur verfügbare Ressourcen werden angezeigt
  + Fehler werden beim Speichern angezeigt
* Header mit Navigationselementen und Logo
* Datenhaltung in MySQL Datenbank (in Azure)
* Simulationsgeschwindigkeit ist anpassbar (für Produktionslinie)
* Simulation läuft im Backend ab und ist unabhängig vom Frontend
* Nur beim Stoppen einer Simulation wird persistiert