FH MÜNSTER ETI FB Elektrotechnik und Informatik
Department of Electrical Engineering
and Computer Science

Entwicklung eines webbasierten Produktkonfigurators für die Modellfabrik

Masterprojekt Wintersemester 2020/2021

Autoren Lucas Weckermann
Lukas van der Stok
Michael Schneider
Niklas Tasler

Abgabedatum 25. Februar 2021

Betreuer Prof. Dr.-Ing. Falk Salewski
Prof. Dr.-Ing. Jürgen te Vrugt
Mirko Venker

Inhaltsverzeichnis

1 Einführung

Dieses Projekt beschäftigt sich mit der Erstellung eines Webshops für die Modellfabrik im Labor für Steuerungstechnik der FH-Münster. Die Modellfabrik besteht aus mehreren Stationen, die miteinander vernetzt sind und zusammen einen Produktionsprozess abbilden.

Die sechs Stationen - Herstellung, Sortierung, Palettierung, Hochregal, Fertigung und Qualitätskontrolle - sind über ein zentrales Förderband miteinander verbunden, was es Ihnen erlaubt Materialien untereinander auszutauschen.

Die Modellfabrik verarbeitet Rohmaterialien - Metallkegel unterschiedlicher Höhe - welche zu verschiedenen Türmen zusammengesetzt werden können, siehe Abbildung 1.1.

Abbildung 1.1: Metallkegel der Modellfabrik

Abgesehen von den drei verschiedenen Höhen, können sich die Metallkegel auch in ihrer Farbe unterscheiden. Es gibt silberne und blaue Metallkegel.

Besonders im Rahmen der sogenannten Industrie 4.0 hat die Individualisierung von Produkten an Bedeutung gewonnen. Nicht nur Autos, sondern auch Schuhe, Müsli und viele weitere Produkte können heute vor der Fertigung nach den Wünschen des Kunden angepasst werden. Vorzugsweise

existiert dazu ein optisch ansprechender Konfigurator im Internet, welcher das individualisierte Produkt anzeigt und die direkte Bestellung ermöglicht.

Vor diesem Hintergrund haben wir für die Modellfabrik einen zeitgemäßen Webkonfigurator entworfen, welcher Nutzern erlaubt Paletten der Modellfabrik mit beliebigen Metallkegeltürmen in Auftrag zu geben. Ein Screenshot unserer Software ist in Kapitel 5 dargestellt.

2 Motivation

Verschiedene Funktionalitäten der Modellfabrik wurden bereits in anderen Masterprojekten umgesetzt. So wurde auch in einem Masterprojekt 2015/2016 ein erster Webshop realisiert. Dieser ist Teil des Onlineauftritts der Modellfabrik, welche mit einem WEBfactory Produkt umgesetzt ist.

Das Unternehmen WEBfactory wurde 1994 als Dienstleister für SCADA-Projekte gegründet. Ein Produkt der Unternehmens ist i4SCADA, eine webbasierte Visualisierungssoftware für SCADA und HMI Anwendungen. Der Webaufritt der Modellfabrik visualisiert die Produktionszustände der einzelnen Stationen der Modellfabrik. Als Erweiterung wurde im Rahmen des Masterprojekts unter anderem ein Unterpunkt *Bestellung* hinzugefügt. Dieser ermöglichte die Konfiguration und Bestellung von individuell konfigurierten Produkten.

Erweiterungen und Änderungen am Produkt i4SCADA führten allerdings dazu, dass der Webshop innerhalb des WEBfactory-Servers nicht länger funktional war. Wir wurden daher mit der Neuentwicklung eines von i4SCADA unabhängigen Webshops beauftragt.

3 Anforderungen

Um Inkompatibilitäten im Rahmen von WEBfactory Updates zu vermeiden, soll der neu erstellte Webshop außerhalb des bestehenden WEBfactory-Servers erstellt werden.

Im folgenden werden zunächst die funktionalen und nicht-funktionalen Anforderungen des neu zu erstellenden Webshops gesammelt, bevor im Anschluss die Entscheidung für eine geeignete Technologie getroffen wird.

3.1 Funktionale Anforderungen

Internetbasierter Auftritt

Kunden aller Welt erwarten heute, dass Sie ihr Produkt im Internet bestellen können. Essentiell für einen Webshop ist es daher, dass dieser öffentlich zugänglich ist. Dazu sollte lediglich ein Webbrowser notwendig sein.

Geräteunabhängig

Neben der öffentlichen Zugänglichkeit muss auch gewährleistet sein, dass der Webshop von beliebigen Endgeräten aufrufbar und benutzbar ist. Einzige Voraussetzung ist das Vorhandensein eines zeitgemäßen Webbrowsers. Die Funktionalitäten des Webshops sollen auf Handys, Tablets und PCs identisch sein.

Konfigurierbare Produkte

Zentrales Merkmal der Industrie 4.0 ist die individuelle Gestaltung und Anpassung von Produkten. Der Webshop muss es Kunden ermöglichen, das Produkt nach eigenen Wünschen zusammenzustellen. Da die Modellfabrik gestapelte Metallkegel unterschiedliche Größe produziert, soll der Kunde jede mögliche Kombination der Kegel als bestellbares Produkt auswählen können.

Anzeige des Lagerbestandes

Heutzutage kann es als Standard angesehen werden, dass Webshops den Lagerbestand eines

Produktes anzeigen. Dies ermöglicht es Kunden abzuschätzen, wann ihre Bestellung ankommen wird. Auf der anderen Seite, kann dies auch die Wahl der Materialien beeinflussen. Gegebenenfalls ändern Kunden ihre Produktkonfiguration, um nur verfügbare Komponenten zu nutzen und so eine schnellere Produktion und Lieferung zu gewährleisten. Der Webshop der Modellfabrik soll dem Kunden anzeigen, welchen Lieferstatus ein konfiguriertes Produkt besitzt.

Anzeige des Bestellstatus

Nach der Bestellung möchte sein Kunde nachvollziehen, in welchem Status sich seine Bestellung befindet. Somit kann der Kunde nachvollziehen, ob die Produktion ausstehend ist, das Produkt sich in der Produktion befindet, oder die Bestellung bereits vollständig verarbeitet wurde. Die Rückmeldung detaillierter Informationen über den Produktionsfortschritt stellt eine moderne Möglichkeit dar, welche die Kundenakzeptanz erhöhen kann. So soll auch der Modellfabrik-Webshop dem Kunden ersichtlich machen, in welchem Stadium der Produktion sich sein konfiguriertes Produkt befindet.

Direkte Kommunikation mit Modellfabrik

Ein weiterer Punkt moderner Produktionsprozesse ist die automatische Verarbeitung von Bestellungen. Dazu soll der Webshop direkt mit der Modellfabrik kommunizieren. Es sollen keine manuellen Schritte notwendig sein, um die Produktion einer Bestellung zu starten oder zu stoppen. Vielmehr sollen alle Bestellungen vollautomatisch ohne notwendige menschliche Interaktion verarbeitet werden. Zu diesem Zweck soll der Webshop direkt mit der Modellfabrik kommunizieren und Bestellungen übertragen.

3.2 Nicht-funktionale Anforderungen

Modernes Erscheinungsbild

Neben der Benutzbarkeit spielt auch das Erscheinungsbild eine immer größere Rolle. Kunden diverser Onlineshops sind moderne Webseiten gewohnt und erwarten ein entsprechend ansprechenden Erscheinungsbild. Um diesem Trend Folge zu leisten, soll auch der Webshop der Modellfabrik moderne Styling-Möglichkeiten nutzen und ein zeitgemäßes Erscheinungsbild liefern.

Wartbarkeit und Erweiterbarkeit

Der Webshop soll leicht wartbar sein und außerdem Erweiterungen zulassen. Änderungen an bestehenden Systemen sind heute Standard und sollen nicht dazu führen, dass ein neuer Webshop erstellt werden muss. Somit kann es etwa sein, dass eine die Modellfabrik in Zukunft

Kegel anderer Größen stapelt oder weitere Farben gewählt werden können. Solche und weitere Änderungen an der Modellfabrik sollen sich im Webshop leicht anpassen lassen.

4 Technologien

Zunächst sollen die verschiedenen Technologien betrachtet werden, welche für die Umsetzung der Anforderungen in Betracht gezogen werden. Die Implementierung des Webshops ist kein neues Problem und es existieren zahlreiche Anbieter für Standardsoftware. Zu den bekannten Shopsystemen gehören etwa *shopify*, *Magento* und *WooCommerce*. Aus dem lokalen Raum ist außerdem das Shopsystem *Shopware* bekannt, welches in Schöppingen entwickelt wird. Dieses soll im folgenden stellvertretend für eine existierende Shopsystem-Lösung betrachtet werden.

4.1 Shopware

Wir haben uns gegen die Benutzung von Shopware entschieden. Trotzdem stellen wir unsere Recherchen zu Shopware in diesem Abschnitt einmal vor.

Die aktuellste Version des Shopsystems ist *Shopware 6*. Dieses bietet etwa Einzel- oder reinen Onlinehändlern die Möglichkeit ihre Produkte leicht online zu verkaufen. Shopware bietet eine Oberfläche, welche es erlaubt selbstständig neue Produkte ohne technische Programmierkenntnisse anzulegen.

Die Standardsoftware hat den Vorteil eines breiten Supports. Viele Mitarbeiter arbeiten täglich an der Weiterentwicklung und Wartung des Shopsystems. Außerdem ist Shopware in einer OpenSource Version verfügbar, was in der Regel für eine große Community sorgt.

Viele große Kunden, darunter Borussia Dortmund, Philips und Frosta, nutzen Shopware für ihre Webshops. Außerdem ist die Software für verschiedenste Endgeräte ausgelegt. Da die Kunden dieser Unternehmen vielfältig sind und mit verschiedensten Endgeräten die jeweiligen Webshops nutzen, ist Shopware entsprechend auf diese Rahmenbedingungen optimiert. Die Oberfläche ist also bereits auf verschiedene Bildschirmgrößen und Auflösungen optimiert.

Neben vielen Features verfolgt Shopware auch einen *API-first* Ansatz. Alle Funktionen den Shops lassen sich also über eine Schnittstelle aufrufen, was eine wichtige Funktionalität für die Kommunikation zwischen Modellfabrik und Webshop darstellen kann.

Eine Vielzahl von Features bringt Shopware in der Standardvariante bereits mit. Mit entsprechenden Programmierkenntnissen ist es allerdings auch möglich Shopware selbst zu erweitern

und entsprechende Plugins zu schreiben. Viele Plugins stehen bereits im *Shopware Store* zur Verfügung und können - teils kostenlos, teils kostenpflichtig - bezogen werden.

Zentrales Feature, welches für den Modellfabrik-Webshop benötigt wird, ist die Unterstützung konfigurierbarer Produkte. Shopware bietet von Haus aus den sogenannten *Variantenkonfigurator* an. Dieser erlaubt es, mehrere Eigenschaften eines Produktes variabel zu gestalten. Dabei kann es sich im Falle von Mode etwa um die Kleidergröße handeln. Der Kunde kann dann beim Einkauf seine Größe selbstständig auswählen. Auch die Kombinationen mehrerer variabler Eigenschaften sind dabei möglich. Shopware generiert automatisch das Kreuzprodukt aller möglichen Varianten eines Produktes.

Eigenschaft	Eigenschafts-Ausprägungen
Anzahl linke Kegel	1, 4, 3, 2
Anzahl rechte Kegel	2, 3, 1
Farbe	Rot, Grün, Blau
Höhe	50 mm, 35 mm, 20 mm

Abbildung 4.1: Konfiguration von Produkteigenschaften in Shopware

Abbildung 4.1 zeigt einen Ausschnitt der Shopware-Administration. Dieser erlaubt die Anlage von Produkteigenschaften. Vorhandene Eigenschaften lassen sich anschließend Produkten zuordnen. Das Beispiel zeigt eine mögliche Kombination von Eigenschaften für den geplanten Konfigurator. In diesem Beispiel wurden vier Eigenschaften angelegt. Der Kunde hat so die Möglichkeit, die Anzahl der Kegel auf der rechten und der linken Seite der Metallplatte. Daneben kann die Farbe und die Höhe der Zylinder variabel ausgewählt werden.

Weist man diese Eigenschaften einem Produkt zu, so ergibt sich im Webshop damit die in Abbildung 4.2 dargestellte Produktansicht für den Kunden. Eine ähnliche Auswahl von Produkteigenschaften ist auch von anderen großen Unternehmen wie *Amazon* bekannt.

Neben den variablen Eigenschaften lassen sich bei Anlage des Produktes noch weitere Merkmale angeben. Dazu gehören typische Angaben wie Versandkosten, Preis, Lagerbestand und die voraussichtliche Versanddauer.

Vorteil dieser Lösung ist ein vollumfänglicher Webshop, welcher alle typischen Funktionen bereits mitbringt. Die Anlage von neuen Produkten und Eigenschaften kann somit ohne Programmier-

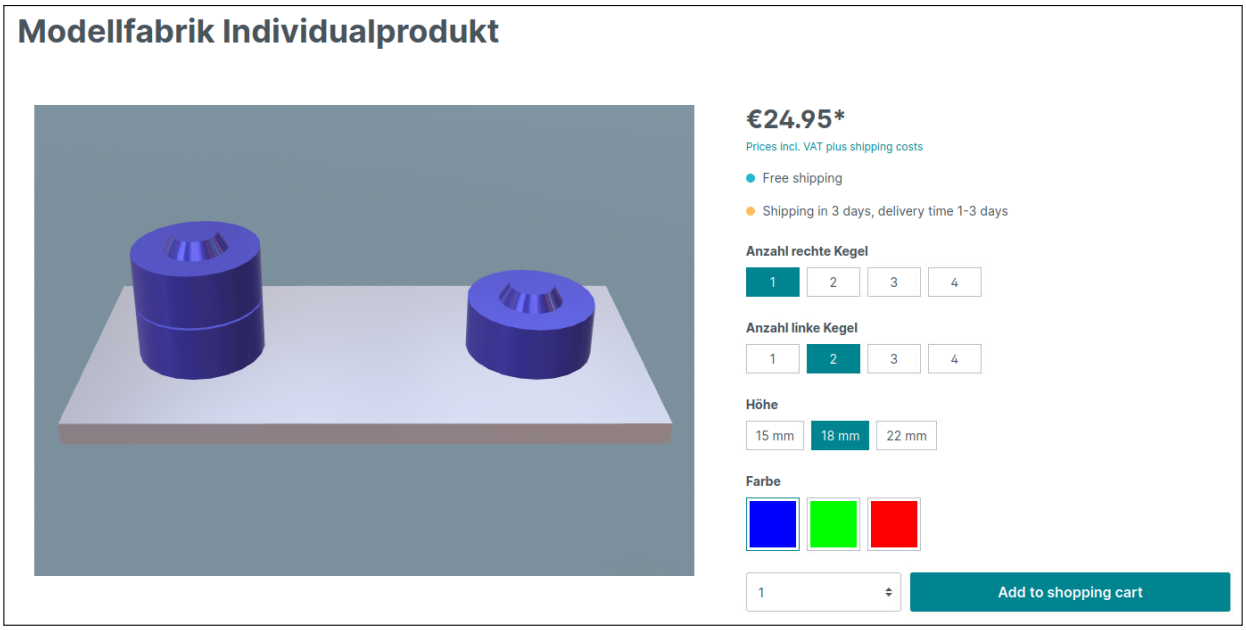

Abbildung 4.2: Produktansicht in Shopware

kenntnisse über eine Administrationsoberfläche bewerkstelligt werden. Auch mit regelmäßigen Updates und einer modernen visuellen Darstellung kann gerechnet werden.

Die Produktansicht zeigt aber auch die Grenzen des Variantenkonfigurators. Im vorliegenden Beispiel können vier Eigenschaften variabel gewählt werden. Trotzdem existieren Einschränkungen bei der Auswahl des Produktes. Alle Kegel haben immer die gleiche Höhe und Farbe. Die Modellfabrik ist allerdings in der Lage Produkte bestehend aus Kegeln verschiedener Höhen und Farben zu erstellen. Möchte man entsprechend alle möglichen Varianten eines Produktes unterstützen, so sorgt dies für eine erhebliche Anzahl an wählbaren Eigenschaften. Soll der Kunde für jede der 8 möglichen Kegelpositionen sowohl Farbe als auch Höhe des Kegels auswählen können, so ergeben sich 24 Produkteigenschaften. Eine solch große Anzahl an auszuwählenden Eigenschaften ist weder zeitgemäß noch kundenfreundlich.

Ein weiterer Aspekt sind die Produktbilder und Preisangaben. Der Variantenkonfigurator generiert aus den verfügbaren Eigenschaften alle möglichen Produktkonfigurationen. Bereits bei nur vier Eigenschaften ergeben sich 144 verschiedene Produktvariationen. Möchte man die den Kunden für jede Konfiguration eine realitätsnahe Vorschau des Produktes anzeigen, so müssen bereits 144 Vorschaubilder erstellt und hinterlegt werden. Erhöht man die Anzahl der Eigenschaften für eine detaillierte Produktkonfiguration, so steigt entsprechend auch der Administrationsaufwand für das Hinterlegen von Produktbildern und weiteren Eigenschaften.

4.2 Eigener Webshop

Als Alternative zu einem existierenden Shopsystem, soll eine selbst entwickelte Lösung in Betracht gezogen werden. Dabei soll vor allem Fokus auf eine variable Produktkonfiguration gelegt werden, da dies den Schwachpunkt bei dem zuvor behandelten Shopsystem darstellt.

Stand der Technik sind bei konfigurierbaren Produkten heutzutage 3D-Konfiguratoren. So kann ein Kunde etwa beim Autokauf verschiedene Lackfarben, Reifen und weitere Ausstattungen auswählen und sich das zu bestellende Auto von allen Seiten im Webbrowser anschauen. Diese basieren auf WebGL. Dabei handelt es sich um eine Schnittstelle für hardwarebeschleunigte 2D und 3D Darstellungen im Webbrowser. Aufbauend auf WebGL existieren einige Bibliotheken, die Erstellung von interaktiven 3D-Szene vereinfachen und eine Abstraktionsschicht darstellen.

Zu den bekanntesten Bibliotheken gehören *Three.js* und *Babylon.js*.

Three.js

Three.js wurde bereits 2010 veröffentlicht. Es gilt als die bekannte Bibliothek für webbasierte 3D-Animationen. Bei Three handelt es sich primär um eine Rendering-Pipeline und weniger um ein Tool für das Erstellen von interaktiven Szenen. Während dies zwar möglich ist, muss etwa für die Nutzung einer Physik-Engine auf externe Pakete wie Physi.js zurückgegriffen werden.

Three.js nutzt für den Großteil des Renderings WebGL, da Three.js allerdings bereits vor WebGL veröffentlicht wurde, kommen für die Darstellung der Szenen auch weitere Technologien zum Einsatz.

Three.js wird auch heute weiterhin aktiv weiterentwickelt und hat eine aktive Community. Die Website enthält viele außerdem viele Beispiele von 3D-Szenen. Die häufigen Änderungen in der Bibliothek führen allerdings auch dazu, dass viele Plugins nicht mehr lauffähig sind. Es sind also in einer bestehenden Anwendung häufig Pflege notwendig, um Änderungen der Schnittstelle zu übernehmen.

Babylon.js

Eine weitere Bibliothek für die Erstellung von 3D-Szenen ist Babylon.js. Dieses wurde 2013 von den beiden Microsoft Mitarbeitern David Catuhe und David Rousset in ihrer Freizeit entwickelt. Inzwischen hat sich Babylon.js zu einer bekannten Bibliothek entwickelt, welche mehr als 190 Unterstützer hat. Neben Microsoft gehören dazu auch weitere bekannte Firmen wie etwa Ubisoft. Babylon.js wird vor allem - aber nicht ausschließlich - für die Entwicklung von webbasierten Spielen genutzt. So nutzt auch die webbasierte Version von *Minecraft* [1] Babylon.js.

[1] `https://classic.minecraft.net`

Ein Vorteil von Babylon.js ist die durch die Entwickler zugesicherte Abwärtskompatibilität. Alle Änderungen und Weiterentwicklungen, die an der Bibliothek vorgenommen werden, führen nicht dazu, dass bestehender Code nicht mehr lauffähig ist. Babylon.js bietet außerdem einige nützliche Tools, welche die Entwicklung von webbasierten Szenen vereinfachen sollen. So extisiert ein Live-Editor - der sogenannte *Playground* - in dem Entwickler Code direkt testen können. Weiter können Shader-Effekte auch ohne tiefgehende Programmierkenntnisse über eine grafische Oberfläche - den sogenannten *Node Material Editor* - zusammengestellt werden. Der zugehörige Code wird automatisch generiert. Auch Tools für ein einfaches Debuggen sind vorhanden.

Der in Betracht gezogene eigene Webshop soll die variable Erstellung eines Produktes erlauben. Dazu sollen Kegel verschiedener Größe und Farbe auf einer Metallplatte palletiert werden können. Für einen solchen Konfigurator sind sowohl Three.js als auch Babylon.js geeignet.

Neben dem eigentlichen Produktkonfigurator werden weitere Komponenten für den Webshop benötigt. Dazu gehört die Anzeige des Bestellstatus und eine Liste aller Bestellungen. Auch für den Bestellvorgang selbst werden Steuerelemente und Dialoge für Rückmeldungen an den Benutzer benötigt. Typischer besteht eine Website aus *HTML, Javascript* und *CSS*. Für die Erstellung von Websites existieren heute drei bekannte Frameworks beziehungsweise Bibliotheken, welche hilfreiche Funktionen mitbringen und die Entwicklung beschleunigen. Diese sind *Angular*, *React* und *Vue*.

Diese bieten eine einfache Möglichkeit eine Website in mehrere Komponenten zu unterteilen. Dies erlaubt eine gute Wartbarkeit und erleichtert Erweiterungen. Ein weiteres zentrales Feature ist die Reaktivität. Eine Änderung an den Daten führt dazu, dass alle abhängigen Daten neu ausgewertet werden. Die Bibliotheken überwachen dabei automatisch den Status der Daten und lösen bei Änderungen automatisch eine Aktualisierung der abhängigen Komponenten und Oberflächenelemente aus.

5 Implementation

5.1 Systemarchitektur

In Abbildung 5.1 ist die Systemarchitektur des Webkonfigurators dargestellt.

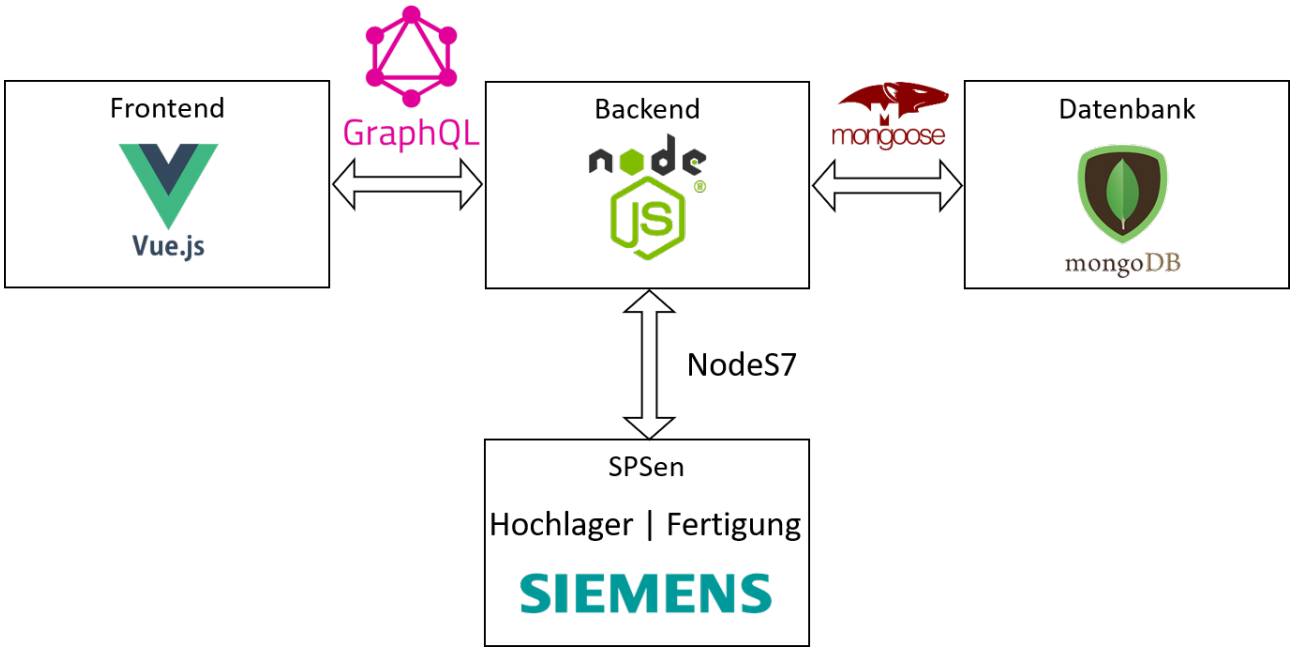

Abbildung 5.1: Systemarchitektur des Webkonfigurators

5.2 Oberfläche

Die Bestellübersicht ist in Abbildung 5.2 dargestellt.

Auf der linken Seite sieht man eine grafische Darstellung der aktuell konfigurierten Palette. Auf der rechten Seite sind die benötigten Metallkegel aufgelistet. Der Lieferstatus besagt, ob ein bestimmter Metallkegel aktuell in der Modellfabrik verfügbar ist. Wir unterscheiden zwischen drei verschiedenen Zuständen, welche durch Ampelfarben dargestellt werden:

1. Grün: Die benötigten Metallkegel sind direkt in der Fertigungsstation vorhanden.

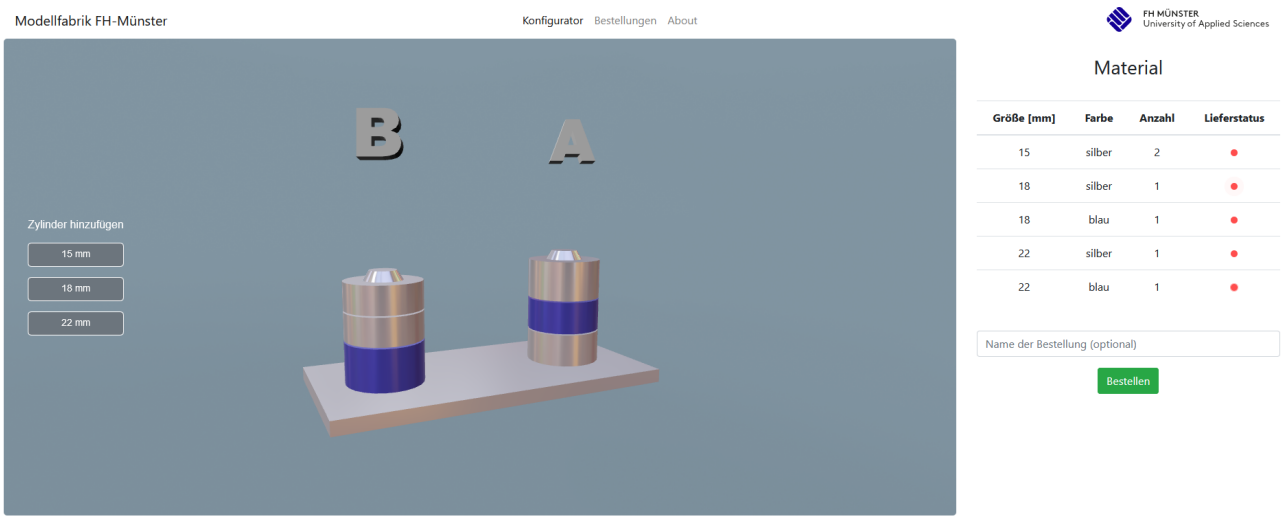

Abbildung 5.2: Bestellübersicht unseres Webkonfigurators

2. Gelb: Die benötigten Metallkegel sind nicht in der Fertigungsstation vorhanden, können allerdings vom Hochregal angefragt werden.

3. Rot: Die benötigten Metallkegel sind weder in der Fertigung noch im Hochlager vorhanden.

Bestellungen, bei denen alle benötigten Metallkegel vorhanden sind, werden der Reihe nach an die Fertigungsstation der Modellfabrik übertragen und dort ausgeführt. Wenn eine Bestellung eine rote Ampel hat, also mindestens ein Bauteil aktuell nicht geliefert werden kann, wird diese nicht zur Fertigungsstation übertragen, da dies sonst die Fertigung blockieren könnte.

Neben der Konfiguration der Paletten bietet unser Konfigurator auch die Möglichkeit den Produktionsstatus von aufgegebenen Bestellungen einzusehen. Dabei kann auch eine Vorschau des bestellten Produktes eingesehen werden.

5.3 Technischer Ablauf einer Bestellung

In der Weboberfläche gibt man eine Bestellung auf. Der Server speichert diese Bestellung in der Datenbank und schreibt sie anschließend in den Datenbaustein der Fertigungs-SPS. Der Name und die Offsets der einzelnen Datenbausteinvariablen müssen in den folgenden Dateien hinterlegt werden:

- /Webshop/server/factory/DatablockProductionSPS.mjs → Für die Fertigungs-SPS
- /Webshop/server/factory/DatablockStorageSPS.mjs → Für die Hochlager-SPS

Vorraussetzung für die Übertragung einer Bestellung ist, dass im Datenbaustein der Fertigungs-SPS der Wert `orderCompleted` auf `true` gesetzt ist. Andernfalls gehen wir davon aus, dass die Modellfabrik aktuell noch mit einer anderen Bestellung beschäftigt ist und warten mit der

Übertragung. Beim erstmaligen Anlegen des Datenbausteins in TIA wird daher empfohlen das Feld mit true zu initialisieren.

Der Server schreibt die Bestellungen in der Reihenfolge, wie sie in der Weboberfläche aufgegeben wurden. Ausnahme bilden Bestellungen, welche basierend auf dem aktuellen Lagerstand der SPS in der Fertigungsstation oder dem Hochlager nicht ausgeführt werden können. Solche Bestellungen werden nicht an die SPS übertragen. Sie werden übersprungen bis die nötigen Materialien in den Lagern vorhanden sind.

Beim Übertragen der Bestellung schreibt der Server in die folgenden Felder des Datenbausteins:

- **orderID**: Bestellungs ID, wird mit jeder Bestellung um 1 hochgezählt. Beginnt nach einem Neustart des Servers wieder bei 0.
- **orderCompleted**: Wird von true auf false gesetzt
- **timeOrderCompleted**: Wird auf den Defaultwert 01.01.1970 gesetzt
- **Turmhöhen** (heightA1 ... heightA5, heightB1 ... heightB5)

Die SPS kann anhand des `orderCompleted` Feldes erkennen, ob eine neue Bestellung angekommen ist. Wenn das Feld auf false gesetzt ist, wurde eine neue Bestellung übertragen.

Die `height` Felder geben die Höhe der Turmbausteine in aufsteigender Reihenfolge an. height1 ist immer der unterste Baustein. Wenn ein Turm weniger als 5 Bausteine benutzt, werden die restlichen `height` Felder vom Server mit 0 beschrieben. Das höchste Bit eines `height`-Werts bestimmt die Farbe des Bausteins 0 = Silber, 1 = Blau.

Sobald die SPS die Bestellung ausgeführt hat und bereit für eine neue Bestellung ist, setzt sie das orderCompleted Feld auf true. Der Server überträgt daraufhin eine neue Bestellung.

5.4 Lagerstand

In regelmäßigen Zeitabständen (aktuell 10 Sekunden) liest der Server den aktuellen Lagerbestand aus dem Datenbaustein der Fertigungsstation und des Hochlagers.

Die SPS sollte die Lagerbestände möglichst aktuell halten, da der Server anhand der Lagerbestände entscheidet, welche Bestellung als nächstes zur SPS übertragen wird.

5.5 Klassenhierarchie

In Abbildung 5.3 ist die Klassenhierarchie des Backend-Servers dargestellt.

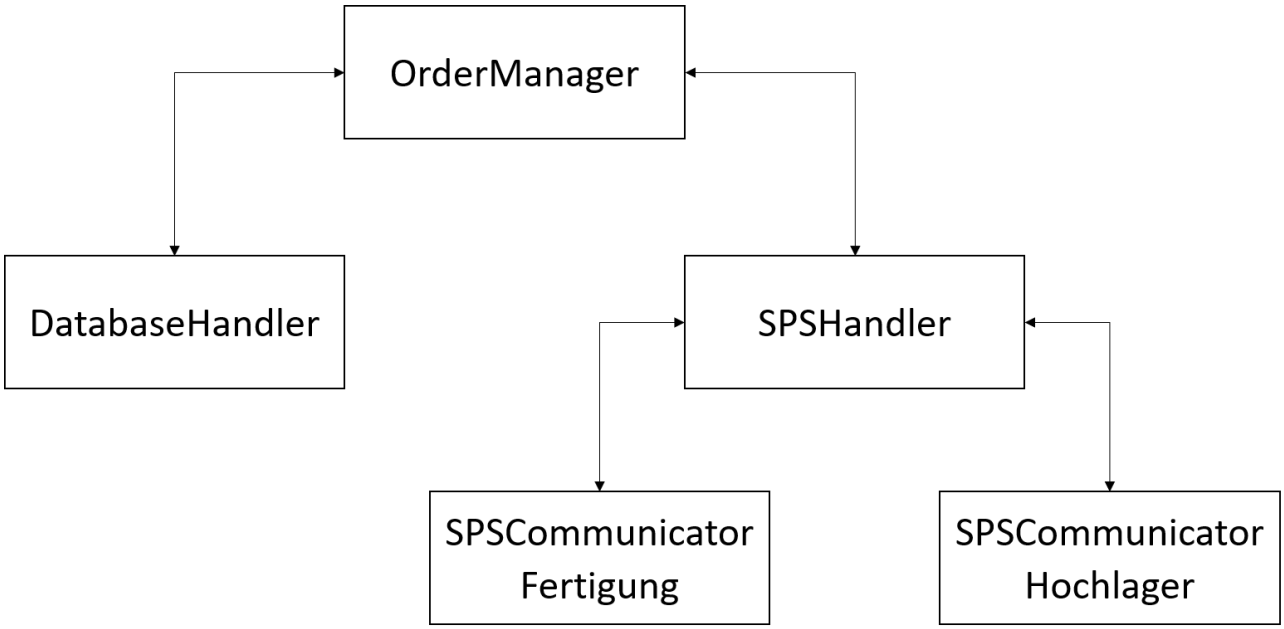

Abbildung 5.3: Klassenhierarchie des Backend-Servers

5.6 Test-Pipeline

In unserem Gitlab Repository ist eine Test-Pipeline wie in Abbildung 5.4 konfiguriert.

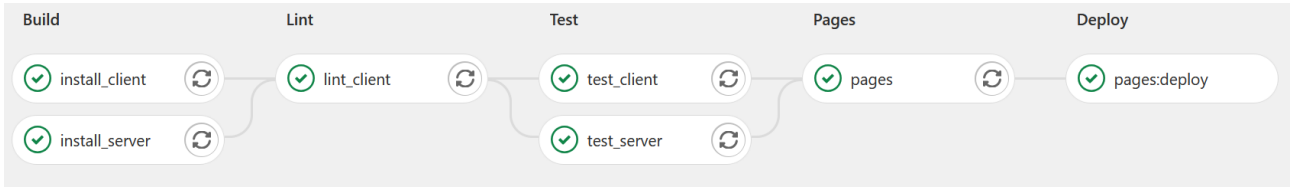

Abbildung 5.4: Screenshot der Arbeitsschritte unserer Pipeline in Gitlab

Diese führt nach jedem Commit alle Tests aus und benachrichtigt den Programmierer, falls eine seiner Änderungen die automatisierten Tests nicht bestanden hat.

6 Webshop Installation

Der Webshop ist über zwei Arten installierbar. Die empfohlene Möglichkeit ist die Installation mit Hilfe von Docker. Die zweite Option ist das manuelle Installieren aller drei benötigten Komponenten.

6.1 Installation mit Docker (Empfohlen)

Docker ist ein Tool, welches die Ausführung aller drei Komponenten - Datenbank, Backend und Frontend - in Containern ermöglicht. Die Ausführung in Containern ermöglicht eine isolierte Ausführung und eine gleichbleibende, reproduzierbare Ausführungsumgebung. Die Installation des Webshops mittels Docker ist sehr einfach, da man lediglich die Dockersoftware manuell installieren muss. Alle anderen benötigten Komponenten werden anschließend automatisch heruntergeladen.

6.1.1 Voraussetzungen

Webshop-Repository

Der Sourcecode des Webshops wird in einem Gitlab-Repository gepflegt. Um den Sourcecode herunterzuladen empfehlen wir die Benutzung von Git. Git kann unter folgendem Link heruntergeladen und installiert werden: `https://git-scm.com/downloads`.
Anschließend gibt man den folgenden Befehl in die Kommandozeile von Linux oder Windows ein, um den Sourcecode aus dem Repository herunterzuladen.

```
# Klonen mit SSH
git clone ssh://git@git.fh-muenster.de:2323/ms180498/
   ↪ masterprojekt-modellfabrik.git

# Klonen mit HTTPS
git clone https://git.fh-muenster.de/ms180498/masterprojekt-
   ↪ modellfabrik.git
```

Docker

Docker stellt die Virtualisierungssoftware dar. Die Installationsanweisung ist verfügbar unter: `https://docs.docker.com/get-docker/`

Docker-Compose

Docker-Compose ist ein Tool für die Definition und Ausführungen von mehreren Containern. Da Datenbank, Backend und Frontend jeweils in einem eigenen Container ausgeführt werden, wird Docker-Compose für die Einrichtung aller drei Container genutzt. Dieses ist erhältlich unter: `https://docs.docker.com/compose/install/`

6.1.2 Datenbank-Konfiguration

Für die Benutzung der Datenbank sind Benutzername und Passwort erforderlich. Um diese Daten zu spezifizieren muss man die `.env.example` Datei im Ordner `/Webshop` duplizieren und anschließend das Duplikat in den Dateinamen `.env` umbenennen. Diese Datei öffnet man nun mit einem Texteditor um den Nutzernamen und das Passwort des Datenbanknutzers festzulegen.

```
# Webshop/.env.example Datei
MONGO_INITDB_ROOT_USERNAME=user
MONGO_INITDB_ROOT_PASSWORD=password
MONGO_INITDB_DATABASE=modellfabrik
```

6.1.3 Backend

Das Vorgehen in Unterabschnitt 6.1.2 wird analog mit der `.env` Datei im Ordner `/Webshop/server` wiederholt. Dort ist eine eigene `.env.example` Datei vorhanden, welche dupliziert und anschließend umbenannt werden muss. Die hier angegebenen Werte für `DB_USER` und `DB_PASSWORD` müssen mit den korrespondieren Werten der `.env` Datei im Verzeichnis `/Webshop` übereinstimmen. Des Weiteren kann man in dieser `.env` Datei die IPs der benutzten SPSen festlegen. Der Wert des `SPS_POLLING_INTERVAL_MS` spezifiziert, in welchem zeitlichen Abstand (Millisekunden) die Datenbausteine der SPSen nach Änderungen abgefragt werden.

```
# Webshop/server/.env.example Datei
DB_URL=mongodb://mongodb:27017/modellfabrik
DB_USER=user
DB_PASSWORD=password

SPS_IP_PRODUCTION=0.0.0.0
```

```
SPS_IP_STORAGE=0.0.0.0
SPS_POLLING_INTERVAL_MS=1000
```

6.1.4 Frontend

Unter `/Webshop/client` erstellen wir die dritte `.env` Datei. Hier duplizieren wir erneut die vorhandene `.env.example` Datei und benennen das Duplikat anschließend in `.env` um.
Diese Datei enthält die Verbindungsinformationen zum Backend. In der Regel können hier die vorgegebenen Werte beibehalten werden. Es sind keine Änderungen notwendig.

```
# Webshop/client/.env.example Datei
VUE_APP_BACKEND_HTTP_LINK=https://backend.server.de:3000/graphql
VUE_APP_BACKEND_WS_LINK=wss://backend.server.de:3000/graphql
```

6.1.5 Deployment

Die Datei `Webshop/docker-compose.yml` enthält alle Informationen, welche Docker-Compose benötigt, um die drei Container zu erstellen. Mit einer geöffneten Kommandozeile im Order `/Webshop` werden die Container über folgenden Befehl gebaut:

```
# Bauen der Container
docker-compose build
```

Nachdem die Container gebaut wurden, können diese gestartet werden:

```
# Starten der Container
docker-compose up -d
```

Das docker-compose Script sorgt dafür, dass alle Container gebaut werden und in der richtigen Reihenfolge (Datenbank → Backend → Frontend) gestartet werden. Der Webshop ist anschließend unter dem Port 8080 verfügbar.

6.1.6 Code-Erweiterungen

Während sich Code-Änderungen im Falle der manuellen Installation (vgl. Abschnitt 6.2) direkt auf die Anwendung auswirken, ist dies mit Docker nicht der Fall. Werden Änderungen am Quellcode vorgenommen, ist es deshalb erforderlich, die Container neu zu erstellen. Folgender Befehl startet die Container und baut diese bei Änderungen an der Codebasis neu. Eventuell laufende Container werden dabei außerdem automatisch gestoppt.

```
# Automatisches stoppen, bauen und neustarten der Container
docker-compose up -d --build
```

6.2 Manuelle Installation

6.2.1 Webshop-Repository

Das Git-Repository mit dem Quellcode muss heruntergeladen werden (vgl. Abschnitt 6.1.1).

6.2.2 Node.js

Für Front- und Backend werden *Node.js* und *npm* benötigt. Bei Node.js handelt es sich um eine plattformunabhängige Laufzeitumgebung für JavaScript. Damit verbunden ist npm ein Paketmanager für die Node.js Umgebung. Node.js ist verfügbar unter:
`https://nodejs.org/`
Mit der Installation von Node.js wird auch npm installiert, sodass keine separate Installation des Paketmanagers notwendig ist.

6.2.3 MongoDB

Zunächst muss MongoDB installiert werden. Dazu wird auf die offizielle Installationsanleitung verwiesen:
`https://docs.mongodb.com/manual/installation/`
Die Backend Komponenten, welche auf die Datenbank zugreifen, gehen davon aus, dass der Zugriff geschützt ist. Somit muss ein Datenbanknutzer mit Administrationsrechten angelegt werden.
Die offizielle MongoDB Anleitung beschreibt das Anlegen eines entsprechenden Nutzers:
`https://docs.mongodb.com/manual/tutorial/enable-authentication/`

6.2.4 Backend

Der Quellcode für das Backend ist im Repository-Ordner `Webshop/server` enthalten. Öffnet man eine Kommandozeile in diesem Ordner, müssen zunächst die benötigten Pakete installiert werden:

```
# Installation der benötigten Pakete
npm install
```

Das Backend baut eine Verbindung zur Datenbank auf und nutzt dabei den Nutzer, welcher in Unterabschnitt 6.2.3 für die MongoDB angelegt wurde. Benutzername und Passwort werden wie in Unterabschnitt 6.1.3 beschrieben hinterlegt.

Der Server kann anschließend gestartet werden. Dazu mit der Kommandozeile in den Ordner `/Webshop/server` navigieren und folgendes Kommando ausführen:

```
# Server starten
npm start
```

Falls man in Zukunft den Server beenden möchte, kann dies durch Betätigen von `ctrl+c` erreicht werden.

6.2.5 Frontend

Der Quellcode für das Frontend ist im Repository-Order `/Webshop/client` enthalten.
Öffnet man eine Kommandozeile in diesem Ordner, müssen - analog zum Backend - zunächst die benötigten Pakete installiert werden:

```
# Installation der benötigten Pakete
npm install
```

Das Frontend kommuniziert über GraphQL mit dem Backend. Die Schnittstelle wird ähnlich wie schon im Falle des Backends über eine `.env` Datei konfiguriert. Der `/Webshop/client` Ordner des Repositories enthält eine `.env.example` Datei, welche die verfügbaren Schlüssel exemplarisch zeigt.

```
# Webshop/client/.env.example Datei
VUE_APP_BACKEND_HTTP_LINK=https://backend.server.de:3000/graphql
VUE_APP_BACKEND_WS_LINK=wss://backend.server.de:3000/graphql
```

Analog zu Unterabschnitt 6.1.4 muss auch hier unter `/Webshop/client` eine neue Datei `.env` angelegt werden, welche die Schnittstellen-URL des Backend Servers enthält. Nur mit der Environment-Datei ist eine Kommunikation zwischen Frontend und Backend möglich.

Beim Frontend handelt es sich im Wesentlichen um eine Sammlung von JavaScript und CSS-Dateien. Um diese an einen Webbrowser auszuliefern, wird ein Webserver benötigt. Je nachdem, ob dieser während der Entwicklung oder produktiv genutzt werden soll, sollte der Server auf unterschiedliche Weise gestartet werden.

Entwicklung

Während der Entwicklung kann ein mitgelieferter Webserver des `vue-cli-service` genutzt werden. Dieser ermöglicht HotSwap - das Ersetzen von Quellcode ohne Neustart des Servers - und ist somit für die schnelle Weiterentwicklung der Anwendung geeignet.

```
# Starten des Servers in Entwicklung
npm run serve
```

Produktion

Für den Einsatz in Produktion sollte der Quellcode vorher optimiert und transpiliert werden.

```
# Bauen der Anwendung für Produktion
npm run build
```

Das Ergebnis ist ein neuer Order `dist`, welcher die Dateien enthält, die von einem Webserver an den aufrufenden Webbrowser ausgeliefert werden.

Der Ordner kann mit einem beliebigen Webserver ausgeliefert werden. Im folgenden wird exemplarisch einmalig ein einfacher HTTP-Server installiert.

```
# Installieren eines HTTP-Servers
npm install -g http-server
```

Zum Ausliefern der Frontend Inhalte kann dann der zuvor installierte Server genutzt werden:

```
# Starten des Frontend-Servers
http-server dist
```

7 Ausblick

Unsere Implementation eines Webshops ist abgeschlossen. Die Umsetzung der folgenden Aufgaben bietet sich für weitere Masterprojekte an:

Implementation Fertigungsstation

Bei der SPS der Fertigungsstation muss implementiert werden, dass diese ein Produkt fertigt, wenn der Webshop-Server eine Konfiguration in den entsprechenden Datenbaustein geschrieben hat. Der Aufbau der Datenbausteine kann dem TIA-Projekt entnommen werden, welches sich im Git-Repository befindet. Die Bedeutung der einzelnen Variablen im Datenbaustein wird in Abschnitt 5.3 erläutert.

An dieser Stelle sei nochmal erwähnt, dass in den Einstellungen der SPSen der GET/PUT Access aktiviert werden muss, damit unser Service auf die Datenbausteine zugreifen kann. Des Weiteren müssen die Variablen in unseren Datenbausteinen über Offsets identifizierbar sein.

Verbindungsstabilität

Unser Server greift in regelmäßigen Zeitabständen auf Datenblöcke des Hochregals und der Fertigungsstation zu. Wenn die Datenblöcke nicht erreichbar sind, sollte dies in der Weboberfläche dargestellt werden.

Der Server sollte nach Verbindungsverlust ohne einen Neustart in der Lage sein, die Verbindungen zu den SPS neu aufzubauen. Dies haben wir bei unserer Implementation berücksichtigt. Allerdings konnten wir dies nicht gründlich testen, da die FH aufgrund von Corona geschlossen war.

Statistikmodul

Anhand der Bestelldaten in der Datenbank lassen sich interessante statistische Auswertungen erstellen. Hier einige Anregungen:

- Durchschnittlich von der Modellfabrik benötigte Zeit für die Erfüllung einer Bestellung → Anhand dieser Statistik könnte man das Lieferdatum einer Bestellung prognostizieren und dem Nutzer anzeigen.

- Am häufigsten bestellte Konfiguration → Diese Konfiguration könnte man dem User als Konfigurationsvorschlag anzeigen

Benutzersteuerung

Jeder Webshop bietet einen Login und eine Benutzersteuerung, welche in in der aktuellen Umsetzung noch fehlen. Unser Webshop kann mit Benutzerprofilen (Login über das Single-Sign-On der FH) erweitert werden, damit man den Nutzern individuelle Statistiken oder Vorschläge zu Bestellungen machen kann. Man könnte unsere Bestellübersicht dann anpassen, sodass Nutzer nur ihre eigenen Bestellungen sehen.

Kosten simulieren

Außerdem können den verschiedenen Bauteilen Kosten zugewiesen und bei der Bestellung der Gesamtpreis berechnet werden. Wenn man eine größere Stückzahl der gleichen Konfiguration bestellt, könnte man einen Mengenrabatt kalkulieren.