



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](#_Toc107911963)

[2. Projektplanung 3](#_Toc107911964)

[2.1 Zeitplanung 3](#_Toc107911965)

[3. Durchführung 4](#_Toc107911966)

[3.2 Design 5](#_Toc107911967)

[3.3 Datenmodell 7](#_Toc107911968)

[3.4 Implementierung 10](#_Toc107911969)

[3.5 Test / Qualitätssicherung 11](#_Toc107911970)

[4. Projektabschluss 11](#_Toc107911971)

[4.1 Soll / IST- Vergleich 11](#_Toc107911972)

[4.2 Fazit 12](#_Toc107911973)

[4.3 Ausblick 12](#_Toc107911974)

[5. Quellenverzeichnis 13](#_Toc107911975)

[5.1 Abbildungsverzeichnis 13](#_Toc107911976)

[5.2 Tabellenverzeichnis 13](#_Toc107911977)

[5.3 Glossar 13](#_Toc107911978)

[6. Anhang 15](#_Toc107911979)

[6.1 Userstories und Anforderungen 15](#_Toc107911980)

[Userstories 15](#_Toc107911981)

[Weitere Anforderungen (Stichpunkte): 16](#_Toc107911982)

# 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:

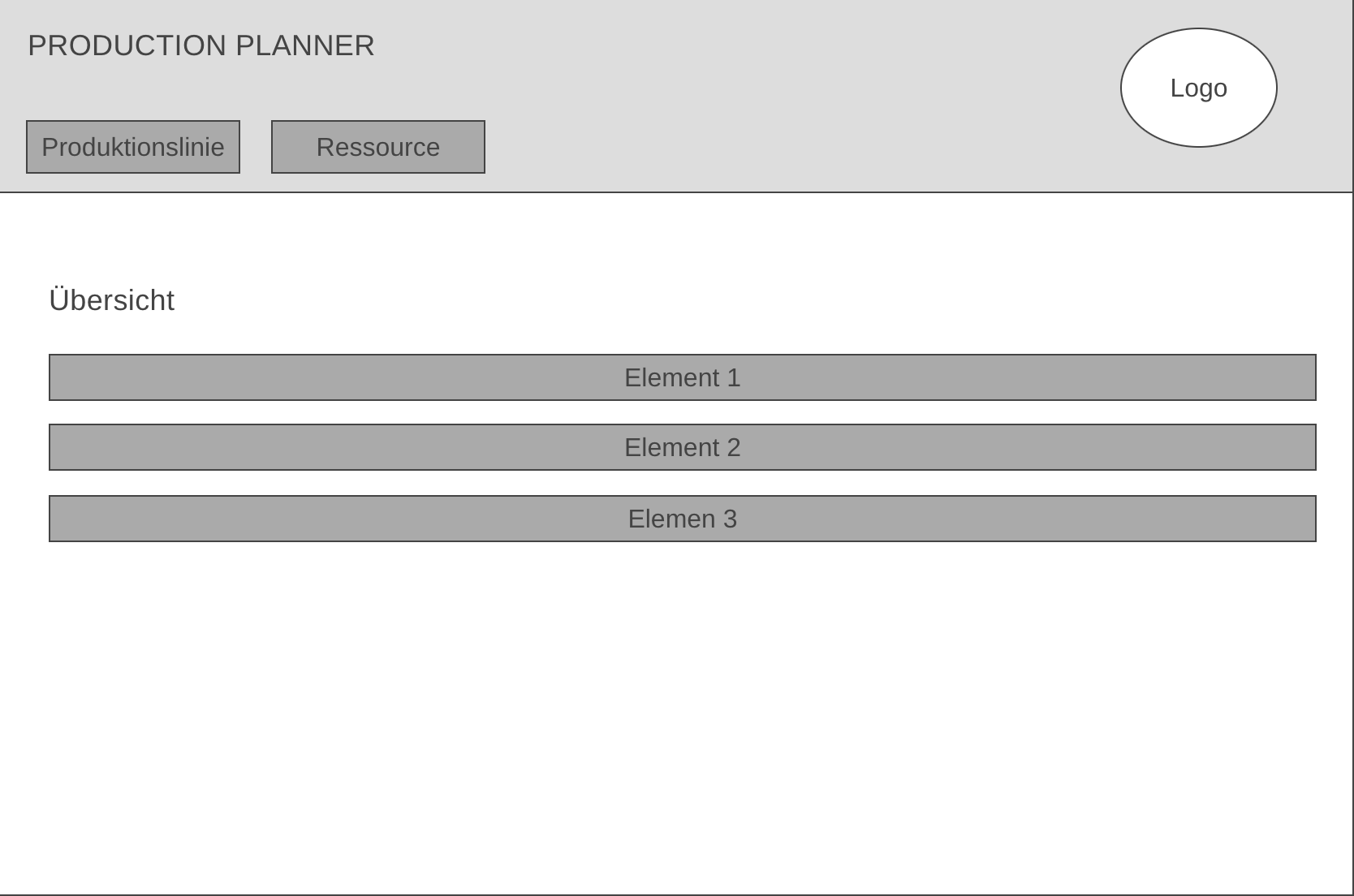
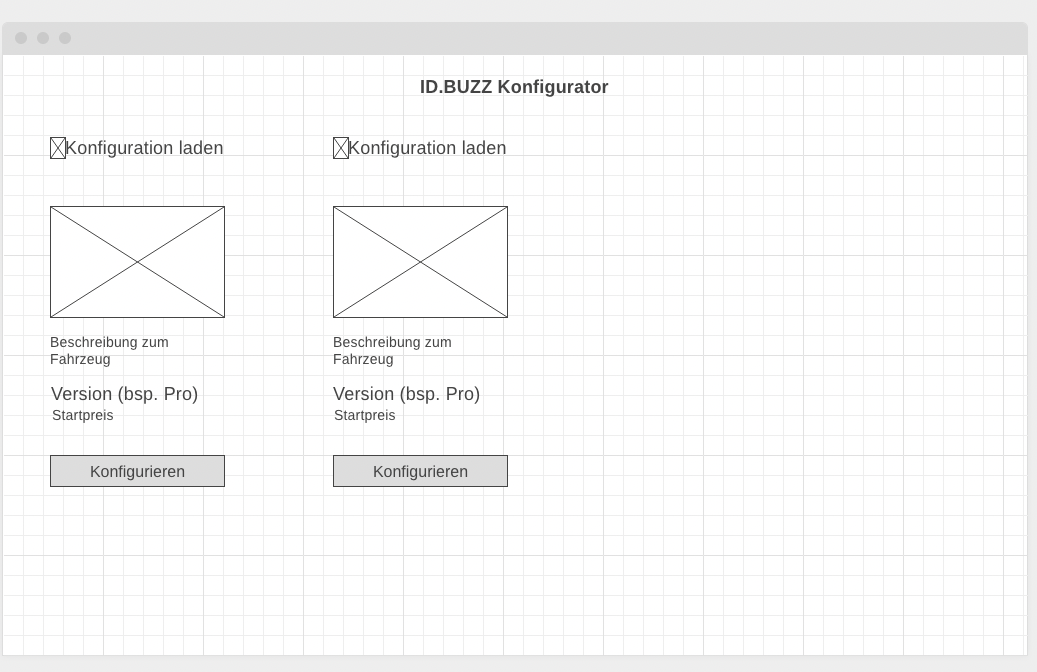
 

Abbildung 1 – Startseite

Die Startseite soll aus einer Kopfzeile bestehen, die dem Nutzer zeigt in welche Applikation er sich befindet und soll zusätzlich die Möglichkeit bieten eine bestehende Konfiguration per eindeutiger Identifikationsnummer zu laden. Darunter soll dem Benutzer eine Reihe an Varianten zur Auswahl geboten werden. Diese beschreiben jeweils eine vorkonfigurierte Ausstattungslinie und können jeweils durch einen Klick auf „Konfigurieren“ individualisiert werden.

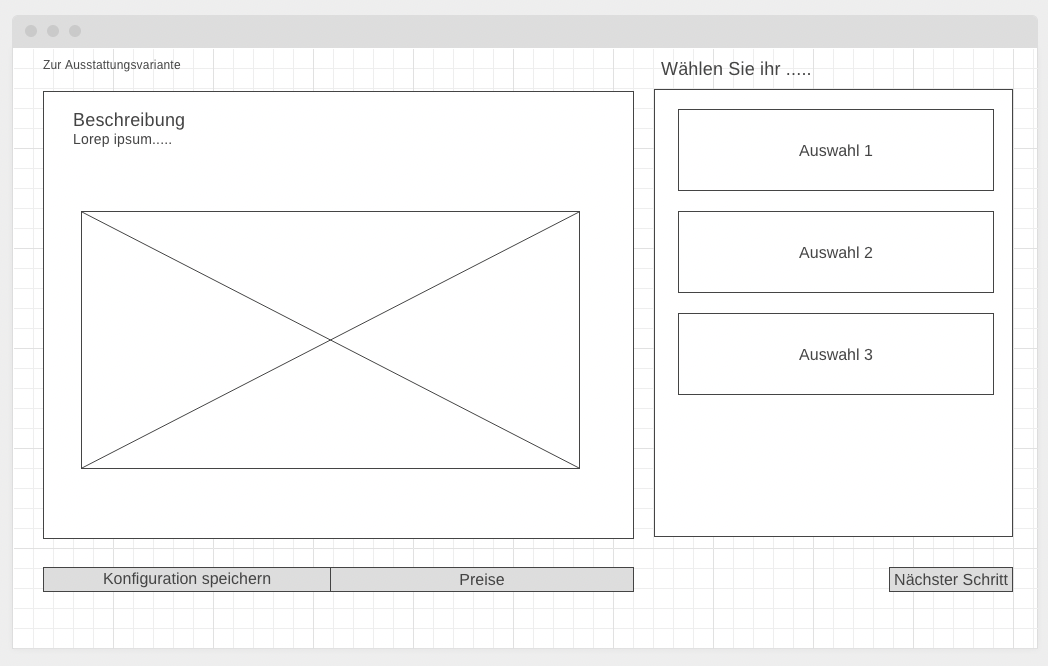


Abbildung 2 - Konfiguration der Komponenten

Nach dem Start einer Konfiguration soll dem Nutzer die gewählte Variante samt Auswahlmöglichkeiten des jeweilige Konfigurationsschritts angezeigt werden. Hierbei wird dem Nutzer jederzeit der aktuell kalkulierte Preis seiner Konfiguration angezeigt und er hat die Möglichkeit die Konfiguration zu jedem Zeitpunkt zu sichern. Diese Ansicht soll nach Möglichkeit für jede Konfigurationskomponente einheitlich aussehen, um eine gute usability zu gewährleisten.

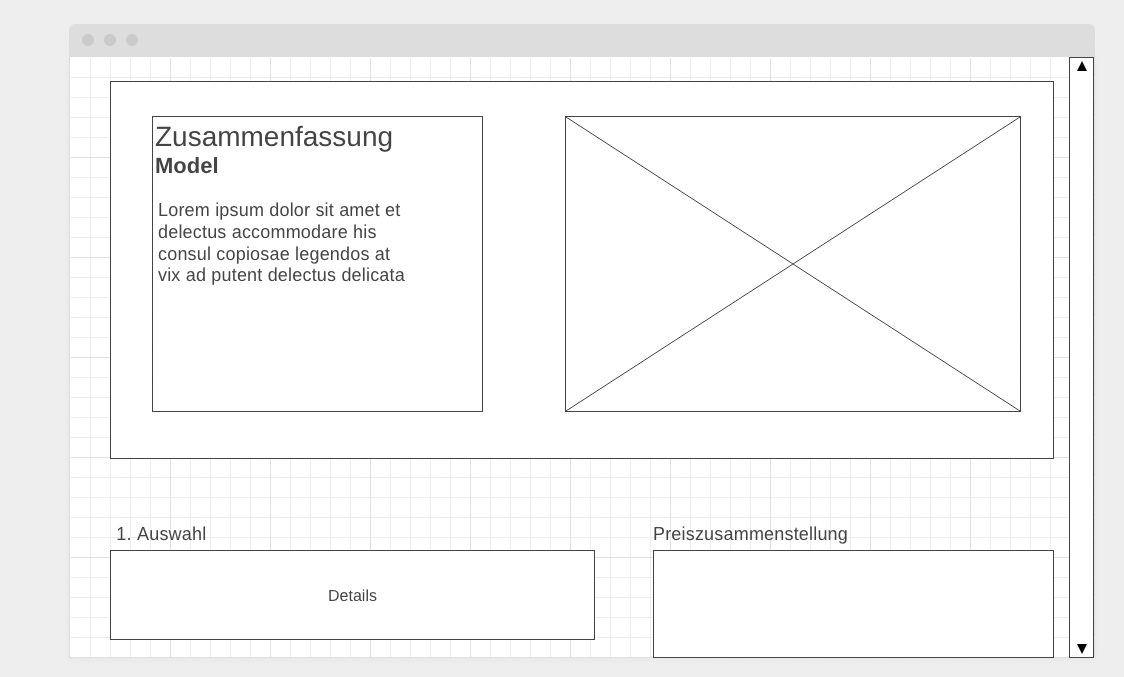


Abbildung 3 - Zusammenfassung

Nach Auswahl der letzten Komponenten wird dem Benutzer eine Zusammenfassung seiner Konfiguration, bestehend aus Variante und Auflistung der gewählten Komponenten, sowie dem berechneten Endpreis angezeigt.

## 3.3 Datenmodell

Für das Datenmodell wurde eine einfache und wiederverwendbare Struktur implementiert, die auf den Klassen „CarConfig“ und „CarConfigAttribute“, sowie dem Enum „Type“ im Backend basiert. Die Klasse CarConfigAttribute steht hierbei für eine Auswahlmöglichkeit (bspw. Motor X, Farbe Y, Felge Z) im Konfigurator und hat folgende Eigenschaften:

|  |  |  |
| --- | --- | --- |
| Name | Datentyp | Zweck |
| id | Long | Eindeutige Identifikation des Attributs |
| title | String | Name des Attributs |
| category | String | Zuordnung einer Unterkategorie |
| description | String | Kurze Beschreibung |
| details | String | Detailierte Beschreibung |
| imgUrl | String | URL des Anzeigebildes |
| thumbnailUrl | String | URL zum Vorschaubild |
| priceInCents | int | Preis in Cent |
| type | Type:Enum | Zuordnung einer Bauteilgruppe |
| exclusions | Set<Long> | Inkompatible Attribute |

Tabelle 2 - Klasse CarConfigAttribute

Das Enum „Type“ beinhaltet die im Konfigurator auswählbaren Bauteilgruppen:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Zweck | Name | Zweck |
| Color | Farbe | Extra | Sonderausstattung |
| Engine | Motor | Interior | Interieur |
| Wheel | Rad und Felge | Model | Variante |

Tabelle 3 - Enum Type

Die Klasse „CarConfig“ dient dem Persistieren von verschiedenen Konfigurationen. Folgende Eigenschaften werden hier verwendet:

|  |  |  |
| --- | --- | --- |
| Name | Datentyp | Zweck |
| id | Long | Eindeutige Identifikation der Konfiguration |
| carConfigAttributes | Set<CarConfigAttribute> | Liste der gewählten Auswahlmöglichkeiten |

Tabelle 4 - Klasse CarConfig

Diese Struktur ermöglicht es alle Objekte gleich zu behandeln und auf einfachstem Wege neue Auswahlmöglichkeiten hinzuzufügen, ohne dafür Anpassungen an dem Quellcode durchführen zu müssen. Dafür werden alle vorhandenen Attribute nach ihrem Typ einem Konfigurationsschritt zugeordnet und dort, anhand ihrer Unterkategorie sortiert.

Hier beispielsweise eine Auswahl des Typen *color* sortiert nach den Kategorien *Metallic* und *Uni/Metallic* (Abbildung 4):

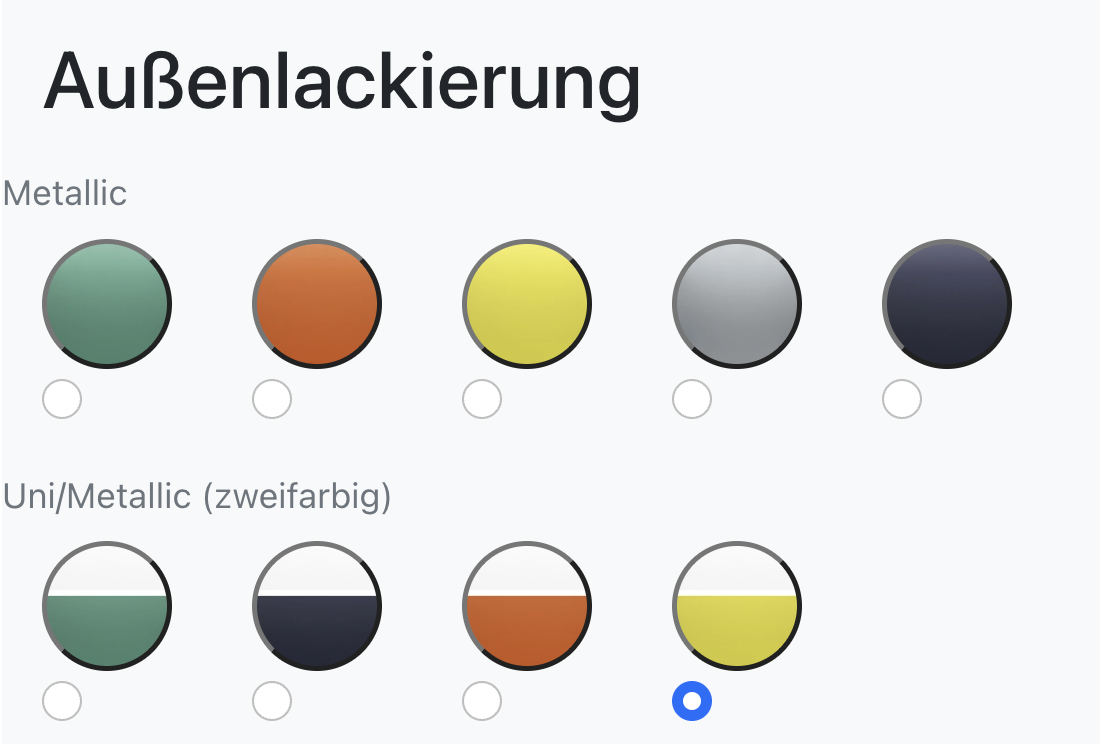


Abbildung 4 - Kategorisierung von Attributen

Durch das gewählte Datenmodell und die verbundene Eindeutigkeit der Attribut IDs könne neue Ausschlusskriterien zwischen den verschiedenen Komponenten definiert werden. Hierzu müssen über die dafür vorgesehene API-Schnittstelle die IDs der sich ausschließenden Attributes angegeben werden, die dann in beiden Attributen als Inkompatiblen Attributen (*exclusions*) ergänzt werden. Im Frontend wird beim Anzeigen der aktuellen Elemente geprüft, ob deren ID sich in den *exclusions* einer der anderen Attribute der Konfiguration befindet. In diesem Fall kann das aktuelle Element nicht mehr ausgewählt werden (siehe Abbildung 5)

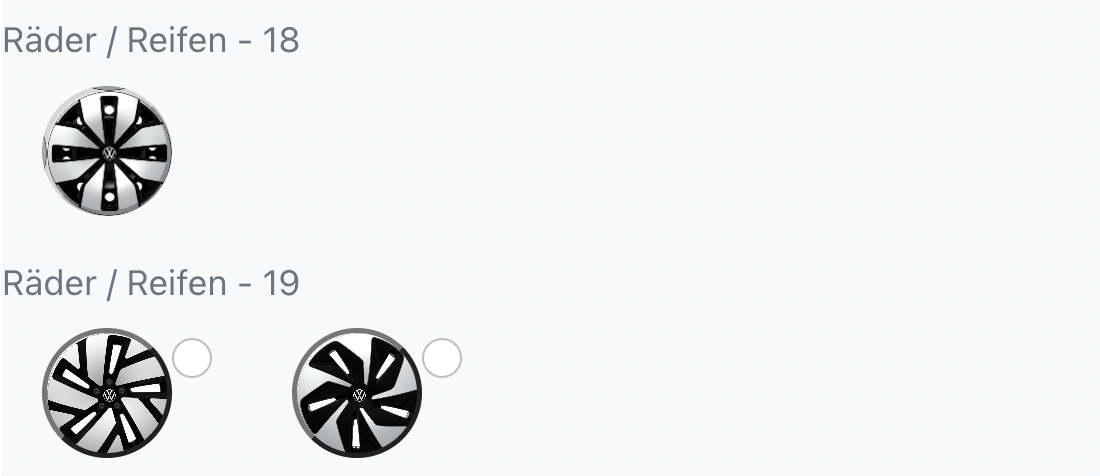


Abbildung 5 - Räder / Reifen - 18 kann nicht ausgewählt werden

Die Bilder wurden für die prototypsche Implementierung direkt im Projektverzeichnisses des Frontends abgelegt und über die Url der Anwendung referenziert. Die Urls wurden mit den einzelnen Attributen als imgUrl bzw. thumbnailUrl in der Datenbank hinterlegt. Für eine produktive Umsetzung können die Bilder so in einem Cloud Blob Storage abgelegt werden und die Urls entsprechend angepasst werden.

Um zu ermöglichen unveränderliche Standartkonfigurationen seitens des Herstellers anzuzeigen, wurden die ersten 50 Konfiguration-IDs reserviert und sind beim Speichern im Backend nicht überschreibbar.

Die gewählte Datenstruktur lässt sich am besten in einem relationalen Datenbankmodell umsetzen. Um Kosten zu sparen und die Erfahrungen der Entwickler bestmöglich zu nutzen, wurde eine Postgres SQL Datenbank verwendet.

Durch die Nutzung von Spring Boot und Spring Data wurden anhand der Annotationen im Quellcode automatisch die Entitäten mit entsprechender Verbindung zueinander erstellt. So konnten möglich Fehlerquellen bei der definition von SQL-Statements ausgeschlossen werden, und der Fokus auf die weitere Logik gelegt werden. Im Endergebnis ergab sich daraus das in Abbildung 6 zu sehende ER-Diagramm.

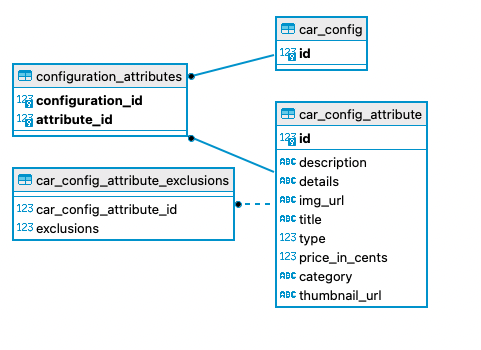


Abbildung 6 - ER-Diagramm

## 3.4 Implementierung

Die Implementierung erfolgte zum Teil remote oder im Mobile Life Campus. Das Vorgehen war jedoch grundsätzlich gleich. Die meiste Zeit arbeiteten wir im Pair- programming. Im Frontend nutzen wir dazu Visual Studio Code als IDE, welches uns mit dem „Liveshare“ Plugin die Möglichkeit bot, den gleichen Code zu sehen und gleichzeitig zu bearbeiten. Für das Backend nutzten wir die Community Edition von IntelliJ. Diese bot mit dem Plugin „Code with me“ zwar ähnliche Funktionalitäten wie Liveshare, hatte jedoch den Nachteil, dass die Sessions auf maximal 30 Minuten begrenzt waren. Deshalb entschieden wir uns hier gemeinsam an einem Rechner zu arbeiten, oder während der Remotearbeit den Bildschirm zu teilen, um so gemeinsam an dem Code arbeiten zu können.

Durch die Arbeit im Pair, war eine ständige Kommunikation gegeben, welche entweder vor Ort, per Microsoft Teams, oder Big Blue Button stattfand. Dadurch konnten Fehler früh erkannt und die Qualität des Codes verbessert werden.

Während der Implementierung achteten wir darauf, regelmäßige und in sich abgeschlossene Commits zu tätigen, um so immer einen funktionierenden Stand der Software zu sichern.

## 3.5 Test / Qualitätssicherung

Um eine möglichst hohe Codequalität sicherzustellen und mögliche Fehler bei der Bereitstellung und Sicherung der Daten auszuschließen wurde der Fokus der Tests auf die Endpunkte des Backends gelegt. Durch die Wahl von Integrationstests konnte die gesamte Datenverarbeitung von der Datenbank bis zur API und umgekehrt abgesichert werden und auf weitere Tests (z.B. Unit Test) verzichtet werden. Um die Tests nicht unnötig aufzublähen und gleichzeitig den vollen Lebenszyklus der Konfigurationsattribute und Konfigurationen durchlaufen zu können wurden diese aufeinander aufbauend definiert und lassen sich nur in der definierten Reihenfolge durchführen.

Um eine gleichbleibende Testumgebung sicherzustellen und nicht mit den produktiven Daten in Konflikt zu kommen, wurden die Tests in einer H2 in-Memory Datenbank durchgeführt, die nach jedem Durchlauf zurückgesetzt wird. Während der gesamten Entwicklung wurde auf Fehler und Bugs geachtet. Diese wurden 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 weitestgehend eingehalten werden. Größere Bugs und Herausforderungen traten kaum auf und hatten somit wenig Einfluss auf die Zeitplanung. Dadurch blieb am Ende des Projekts genug Zeit um den Code abschließend zu strukturieren und zu optimieren. Bei dem Entwurf der grafischen Umsetzung gab es Abweichungen zu der finalen Umsetzung. Diese haben jedoch keine Auswirkungen auf die im Projektantrag gestellten Anforderungen, welche vollständig abgedeckt werden konnten.

## 4.2 Fazit

Trotz der kurzen Projektzeit und den begrenzten Ressourcen, konnten alle Anforderungen an die Software implementiert werden. Darüber hinaus konnten alle Funktion zur Zufriedenheit der Entwickler umgesetzt werden und es musste nicht auf rudimentäre Umsetzungen zurückgegriffen werden. Die Zusammenarbeit im Team lief äußerst produktiv und beide Teammitglieder konnten sich gegenseitig Unterstützen und voneinander lernen. Gemeinsam konnten komplexe Funktionen und Strukturen implementiert werden, von denen beide Teammitglieder lernen und profitieren konnten. Die anfängliche Planung der Struktur erwies sich als sehr hilfreich für den weiteren Projektverlauf, auch wenn die finale Implementierung davon abweicht.

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

Trotz Erfüllung aller Anforderung aus dem Projektantrag, hat die Software noch Potenzial verbessert und um weitere Funktionen erweitert zu werden. Hierzu zählen die Darstellung bestimmter Komponenten, oder das Einfügen verschiedene Ansichten der gewählten Konfiguration (z.B. bei der Räderauswahl). Daneben wäre es für eine produktive Applikation sinnvoll die Bilder in einem seperatem Blob Storage zu halten, um das Frontend schlank zu halten. Auch sollte es dem Nutzer möglich sein eine abgeschlossene Konfiguration zu bestellen, um so den größtmöglichen Nutzen zu erzielen.

# 

# 5. Quellenverzeichnis

## 5.1 Abbildungsverzeichnis

[Abbildung 1 – Startseite 5](#_Toc107922760)

[Abbildung 2 - Konfiguration der Komponenten 6](#_Toc107922761)

[Abbildung 3 - Zusammenfassung 6](#_Toc107922762)

[Abbildung 4 - Kategorisierung von Attributen 8](#_Toc107922763)

[Abbildung 5 - Räder / Reifen - 18 kann nicht ausgewählt werden 9](#_Toc107922764)

[Abbildung 6 - ER-Diagramm 10](#_Toc107922765)

## 5.2 Tabellenverzeichnis

[Tabelle 1: Grobe Zeitplanung 4](#_Toc107922769)

[Tabelle 2 - Klasse CarConfigAttribute 7](#_Toc107922770)

[Tabelle 3 - Enum Type 7](#_Toc107922771)

[Tabelle 4 - Klasse CarConfig 8](#_Toc107922772)

## 5.3 Glossar

API Application Programming Interface

Bug Programmfehler

Bug Fixing Beheben von Programmfehlern

Java Klassen Dient der Erzeugung von Objekten in Java

MVC Pattern Model-View-Controller Pattern.

Pairprogramming Zwei Entwickler programmieren im Vier-Augen-Prinzip.

Git- 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.

Trello- Board Online Kanban Board

# 6. Anhang

## 6.1 Userstories und Anforderungen

### Userstories

#### Userstory 1:

Als Anwender möchte ich, wenn ich die Startseite aufrufe, eine Auswahl über verfügbare Basis Modellvarianten angezeigt bekommen.

#### Akzeptanzkriterien:

* Es werden drei Modellvarianten angezeigt.
* Es wird eine Beschreibung und ein Grundpreis zu jeder Variante angezeigt.
* Es wird jeweils ein Button zur weiteren Konfiguration angezeigt.

#### Userstory 2:

Als Anwender möchte ich während des Konfigurationsvorgangs jederzeit den Preis meiner aktuellen Konfiguration einsehen können.

Akzeptanzkriterien:

* Der Preis wird in jedem Schritt der Konfiguration angezeigt.

#### Userstory 3:

Als Anwender möchte ich, wenn ich den Konfigurationsvorgang komplett durchlaufen habe, eine Übersicht über meine Konfiguration angezeigt bekommen, die ich in den Formaten PDF und TXT ausdrucken kann.

#### Akzeptanzkriterien:

* Zum Abschluss der Konfiguration wird eine Zusammenfassung der Konfiguration angezeigt
* Die Zusammenfassung lässt sich als PDF Datei speichern
* Die Zusammenfassung lässt sich als TXT Datei speichern

### Weitere Anforderungen (Stichpunkte):

* Konfigurationsdialog mit mehreren Stufen
* Einzelne Stufen werden dynamisch aus den Konfigurationsattributen erstellt (Kategorisiert nach Typ und Kategorie)
* Ausgewählte Daten werden zwischengespeichert
* Datenhaltung in Postgres Datenbank (in Azure)
* Einschränkungen können jederzeit zwischen zwei Komponenten erstellt werden (Komponenten schließen sich gegenseitig aus)
* Umsetzung Stufe Interieur
* Umsetzung Stufe Extras (inkl. Exterieur)
* Umsetzung Stufe Felge
* Umsetzung Stufe Farbe
* Umsetzung Stufe Motorisierung
* Nicht wählbare Optionen werden gekennzeichnet
* Gültige Konfiguration lässt sich jederzeit speichern
* Beim Speichern wird ein Code angezeigt mit dem die Konfiguration wieder geladen werden kann
* Geladene Konfigurationen werden unter dem selben Code gespeichert (wenn sie gespeichert werden)