

# Entwicklung und Implementierung eines plattformübergreifenden mobilen Asset- Management-Systems für das „Da Vinci“-Projekt

Bachelor-Arbeit

Hochschule Bielefeld

University of Applied Sciences and Arts

Nataliia Amanzhayeva

geboren am 25.08.1887 in Busk, Ukraine

27. Juli 2024

# Zusammenfassung

Diese Arbeit beschreibt die Entwicklung und Implementierung einer Applikation in Form eines plattformübergreifenden mobilen Asset-Management-Systems für das „Da Vinci“-Projekt. Das „Da Vinci“-Projekt bietet Einblicke in die Symbiose aus Kunst und Technik und illustriert, wie Leonardo da Vinci beide Disziplinen zu Lebzeiten und darüber hinaus prägte.

Das Ablegen und die Pflege der Kunstwerke sind mit viel Aufwand verbunden, da sich viele Kunstwerke aus mehreren komplexen Elementen, wie bspw. Sockel, diversen Accessoires und spezieller Verpackung zusammensetzen können.

Das mobile Asset-Management-System unterstützt Projektmitarbeitende, indem eine digitale Grundlage zum Ablegen von Kunstwerken und Zubehör geschaffen wird. Hierzu wurde eine Datenbank als Ablage für die Daten implementiert. Die Daten innerhalb der Datenbank werden durch das Personal über ein mobiles Endgerät gepflegt. Zusätzlich wurden bereits vorbereitende Maßnahmen für den Scan per QR-Code realisiert, um in Zukunft einen leichten Informationsabruft zu ermöglichen.

Im Anschluss an die Beschreibung der Entwicklung und Implementierung folgt eine Evaluierung per Akzeptanztests in Form von Black-Box-Tests. Um entsprechende Kriterien prüfen zu können, wurden die Anforderungen an die Anwendung zuvor in semi-strukturierten Interviews erfasst.

Die Ergebnisse der Arbeit zeigen, dass die Anwendung über eine intuitive Benutzerführung verfügt. Projektmitarbeitende fanden einen schnellen Anschluss in der Bedienung, da die Anwendung klar und strukturiert die internen Geschäftsprozesse abbildet.

In nachfolgenden Arbeiten soll das Scannen per QR-Code von Kunstwerken als weitere Funktion implementiert und evaluiert werden. Darüber hinaus sollten die Projektmitarbeitenden über einen definierten Zeitraum bei Nutzung der Anwendung begleitet werden. So kann sichergestellt werden, dass die Anwendung über einen längeren Zeitraum robust eingesetzt werden kann und so wird die Erfassung von zusätzlichen oder sich ändernden Anforderungen ermöglicht.

10/2023

## I. Eigenständigkeitserklärung\*

*Declaration of originality\**

Hiermit versichere ich  
Hereby, I

**Amanzhayeva Nataliia**

Name, Vorname

Last name, First name

**1126806**

Matrikelnummer

Student ID number

**Ingenieurinformatik**

Studiengang

Study programme

dass ich die vorliegende **Bachelorarbeit / bachelor thesis**  
*affirm that I have prepared the present*

(bei Gruppenarbeit mein bearbeiteter Teil) mit dem Thema  
*(in case of group work the part I have prepared) with the topic*

Klicken oder tippen Sie hier, um Text einzugeben.

Entwicklung und Implementierung eines plattformübergreifenden mobilen Asset-Management-Systems für das "Da Vinci"-Projekt.

selbstständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen – einschließlich Tabellen, Karten, Abbildungen etc. –, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Werken und Quellen (dazu zählen auch Internetquellen) entnommen wurden, sind in jedem einzelnen Fall mit exakter Quellenangabe kenntlich gemacht worden.

*independently and without using any other than the indicated aids. All passages – including tables, maps, figures, etc. – taken verbatim or rephrased from published and unpublished works and sources (including Internet sources) have been identified in each individual case with exact reference to the source.*

Zusätzlich versichere ich, dass ich beim Einsatz von generativen IT-/KI-Werkzeugen (z. B. ChatGPT, BARD, Dall-E oder Stable Diffusion) diese Werkzeuge in einer Rubrik „Übersicht verwendeter Hilfsmittel“ mit ihrem Produktnamen, der Zugriffsquelle (z. B. URL) und Angaben zu genutzten Funktionen der Software sowie Nutzungsumfang vollständig angeführt habe. Wörtliche sowie paraphrasierende Übernahmen aus Ergebnissen dieser Werkzeuge habe ich analog zu anderen Quellenangaben gekennzeichnet.

*In addition, I assure that, when using generative IT/AI tools (e.g., ChatGPT, BARD, Dall-E, Stable Diffusion), I have listed these tools in full in a section "Overview of tools used" with their product name, the access source (e.g., the URL) and information on the functions of the software used as well as the scope of use. I have marked verbatim and paraphrased quotes from the results of these tools in the same way as I have marked other sources.*

**Mir ist bekannt, dass es sich bei einem Plagiat um eine Täuschung handelt, die gemäß der Prüfungsordnung sanktioniert werden wird.**

*I am aware that plagiarism is a form of cheating that will be penalised according to the examination regulations.*

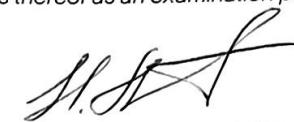
**Ich versichere, dass ich die vorliegende Arbeit oder Teile daraus nicht bereits anderweitig innerhalb oder außerhalb der Hochschule als Prüfungsleistung eingereicht habe.**

10/2023

*I certify that I have not already submitted the present work or parts thereof as an examination performance elsewhere within or outside the university.*

Bielefeld, 26.08.2024

Ort, Datum  
Place, date



Unterschrift  
Signature

\* Bitte legen Sie diese Eigenständigkeitserklärung ausgefüllt und unterzeichnet Ihrer Arbeit am Ende bei. Sollte diese fehlen, wird die Arbeit nicht korrigiert bzw. bei endgültiger Nichtvorlage als Täuschungsversuch gewertet.

\* Please complete and sign this declaration of originality and enclose it with your work at the end. If this is missing, the work will not be evaluated or, in case of final non-submission, it will be considered an attempt to cheat.

# Inhaltsverzeichnis

1	Einleitung .....	10
2	Definitionen .....	11
3	Stand der Technik .....	13
3.1	Mobile Datenerfassung .....	13
3.2	App Entwicklung.....	14
4	Anforderungen .....	16
4.1	Anwendungsfalldiagramm .....	17
4.2	Lastenheft .....	18
4.2.1	Zielbestimmung.....	18
4.2.2	Produkteinsatz .....	18
4.2.3	Produktfunktionen .....	18
4.2.4	Produktdaten.....	19
4.2.5	Produktleistungen .....	19
4.2.6	Qualitätsanforderungen.....	20
4.3	Pflichtenheft.....	20
4.3.1	Zielbestimmung.....	20
4.3.2	Produkteinsatz .....	21
4.3.3	Produktübersicht .....	22
4.3.4	Produktfunktionen .....	22
4.3.5	Produktdaten.....	34
4.3.6	Produktleistungen .....	34
4.3.7	Qualitätsanforderungen.....	36
4.3.8	Benutzeroberfläche .....	36
4.3.9	Nichtfunktionale Anforderungen .....	38
4.3.10	Technische Produktumgebung .....	38
5	Konzept.....	38

5.1	Architektur .....	38
5.2	Frontend .....	40
5.3	API .....	41
5.4	Backend .....	42
6	Implementierung .....	42
6.1	Technische Grundlagen .....	42
6.2	Flutter .....	44
6.2.1	Vergleich und Begründung .....	45
6.2.2	Einführung .....	46
6.2.3	Struktur .....	47
6.2.4	Design der Benutzeroberfläche der Anwendung .....	48
6.2.5	HTTP-Anfragen an die API .....	49
6.2.6	Navigation in der Anwendung .....	51
6.2.7	Produktfunktionen .....	52
6.3	Progressive-Web-App .....	56
6.3.1	QR-Code und Kameraverwendung .....	56
6.3.2	Geolokalisierung .....	57
6.4	MongoDB .....	59
6.4.1	Vergleich von Datenbanktechnologien .....	59
6.4.2	Aufbau .....	60
6.4.3	Umsetzung der Datenbank .....	62
6.5	REST-API .....	66
6.5.1	Vergleich API-Technologien .....	66
6.5.2	Einführung .....	68
6.5.3	Express .....	69
6.5.4	Initialisierung des Servers .....	70
6.5.5	Authentifizierung und Autorisierung .....	71
6.5.6	Routing .....	74

6.5.7	Fehlerbehandlung .....	75
6.5.8	Dateimanagement.....	77
7	Evaluierung .....	77
7.1	Technische Spezifikationen der Testumgebung.....	77
7.2	Akzeptanztest.....	78
7.3	Performance-Test.....	79
8	Fazit .....	79
8.1	Limitierungen.....	80
8.2	Ausblick.....	80

# Abkürzungsverzeichnis

ACID *Atomicity, Consistency, Isolation, Durability*

AMS *Asset-Management-System, Asset-Management-System*

API Application Programming Interface

App *Application*

BASE Basically Available, Soft State, Eventual Consistency

BSON Binary JSON

CORS Cross-Origin Resource Sharing

CSS *Cascading Style Sheets*

DBMS *Datenbankmanagementsystem, Datenbankmanagementsystem*

ER-Modell *Entity-Relationship-Modell*

GPS *Das Global Positioning System*

HTML *Hypertext Markup Language*

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

JWT *JSON Web Token*

NoSQL Not only SQL

PWA *Progressive-Web-App*

REST-API Representational State Transfer API, Representational State Transfer API

SOAP-API *Simple Object Access Protocol API*

SQL *Structured Query Language*

URL Uniform Resource Locator

VSCode *Visual Studio Code*

# Abbildungsverzeichnis

Abb. 1: Anwendungsfalldiagramm des mobilen Asset-Management-Systems im Da Vinci-Projekt .....	17
Abb. 2: Anwendungsfalldiagramm des Asset-Management-Systems im Da Vinci-Projekt. Admin steht für Administrator. WMA bedeutet wissenschaftliche Hilfskraft und SWA studentische Hilfskraft.....	22
Abb. 3: Systemarchitektur.....	39
Abb. 4: Hauptfenster nach der Anmeldung im System. Ansicht auf dem Desktop-PC. .....	49
Abb. 5: Hauptfenster nach der Anmeldung im System. Ansicht auf dem Handy.....	49
Abb. 6:Hauptansicht für Gestaltungselemente. Ansicht auf dem Desktop-PC.....	49
Abb. 7: Hauptansicht für Gestaltungselemente. Ansicht auf dem Handy.....	49
Abb. 8: Navigationssystem der Anwendung im Frontend. ....	51
Abb. 9: Benutzerverwaltung Admin.....	53
Abb. 10: Benutzerverwaltung. SWA und WMA .....	53
Abb. 11: Dialogfenster zur Erstellung von einem neuen Vermögenswert. ....	53
Abb. 12: Ausschnitt der Detailansicht eines mechanischen Modells. ....	53
Abb. 13: Benutzeroberfläche für die Erstellung eines Angebots der Kunde.....	55
Abb. 14: Erweiterung des Angebots um mechanische Modelle .....	55
Abb. 15: Angebot des Kunden, nachdem die Vermögenswerte zur Angebotsliste hinzugefügt wurden. Das System ist bereit die Angebotsliste zu speichern und wartet auf die Aktion des Benutzers. ....	55
Abb. 16: Das Angebot des Kunden ist erstellt. Eine neue Ausstellung kann angelegt werden.....	55
Abb. 17: Das Dialogfenster, um eine neue Ausstellung zu erstellen.....	55
Abb. 18: Dialogfenster, um eine neue Ausstellung zu erstellen, nachdem der Kunde ausgewählt wurde.....	55
Abb. 19: Zugriff auf die Kamera.....	57
Abb. 20: Kamera in aktivem Zustand und ist für die Scannung der QR-Codes bereit. .....	57
Abb. 21: Standortüberwachung des Benutzers. ....	58
Abb. 22: Struktur der implementierten Datenbank.....	61

Abb. 23: Entity-Relationship-Modell.....	63
Abb. 24: Dokumentstruktur eines Vermögenswerts. Die gelben Pfeile zeigen auf eingebettete Daten, die blaue auf Referenzen. ....	64
Abb. 25:Login Sequenzdiagramm .....	73
Abb. 26: Von Entwicklern weltweit verwendete Cross-Platform mobile Frameworks 2019-2023 [32] .....	84
Abb. 27: Verteilung der beliebtesten Datenbankmanagementsysteme weltweit im Jahr 2022 [32].....	85
Abb. 28: Anmeldefenster. Mobile Ansicht.....	85
Abb. 29: Hauptfensteransicht bei Tablet- und Desktopansicht.....	86
Abb. 30: Zeit der Ausführung der Datenbankoperationen. Die erste Spalte steht für "abrufen", die zweite für "speichern", die dritte für "aktualisieren". .....	86
Abb. 31: Entity-Relationship-Modell des Systems. ....	87
Abb. 32: Aktivitätsdiagramm zu dem Erstellen eines Angebots für einen Kunde.....	97
Abb. 33: Aktivitätsdiagramm zum Erstellen eine Ausstellung.....	98

## Tabellenverzeichnis

Tab. 1: Vergleich von App-Kategorien.....	16
Tab. 2: Statuscodes der implementierten API .....	76

# 1 Einleitung

Der digitale Wandel verändert den Arbeitsalltag vieler Branchen. Auch die Kunstbranche wird aktuell durch innovative Technologien beeinflusst. Drei Beispiele hierfür sind: Non-Fungible Tokens (NFT), die dazu verwendet den Besitzer eines Objekts zu referenzieren [1], die Nutzung von künstlicher Intelligenz, um neue Bilder aus einer Datenbasis zu generieren [2] oder das Tracking von Ereignissen, bspw. um Lieferketten nachvollziehen zu können [3, 4].

Originale Kunstwerke bieten jedoch die Basis für innovative Technologien dieser Art, so ist die Verknüpfung von Kunst und Technologie, wie Leonardo Da Vinci sie einst lebte, heute noch spürbar. Diese Arbeit beschäftigt sich mit der Pflege von Kunstwerken und weiteren Vermögenswerten. Damit eine Grundlage für die digitale Pflege von Vermögenswerten geschaffen werden kann, soll in dieser Arbeit ein Asset-Management-System (AMS) entwickelt werden. Zusätzlich soll das AMS mobil einsetzbar sein, wobei *mobil* in diesem Sinne zweideutig ist. Zum einen sind die erfassten Vermögenswerte mobile Gegenstände, wie bspw. Kunstwerke und Zubehör. Und zum anderen soll das System sowohl für Desktop-PCs als auch für Laptops und Smartphones ausgelegt sein. Eine weitere Kernanforderung ist, dass das System plattformübergreifend betrieben werden soll, also auf mehreren Betriebssystemen wie Windows, Android und iOS funktioniert. Somit ist diese Arbeit dem Bereich von mobilen Anwendungen zuzuordnen [5, 6].

Das Ziel dieser Arbeit ist eine Grundlage für die digitale Erfassung von Vermögenswerten zu schaffen. Aufbauend auf dieser Grundlage können in Zukunft verschiedene Technologien wie NFT, künstliche Intelligenz oder ein Ereignistracking etabliert werden.

Im Laufe der Arbeit werden zuerst theoretische Grundlagen präsentiert, anschließend folgt eine Analyse des Stands der Technik, danach werden Anforderungen an das AMS dargestellt, darauf folgt eine detaillierte Beschreibung der Systemarchitektur und der Implementierung, danach wird das AMS evaluiert und im Abschluss werden die Ergebnisse dieser Arbeit zusammengefasst.

## 2 Definitionen

Im Folgenden werden die zentralen Begriffe und Definitionen erörtert, die für das Verständnis der vorliegenden Arbeit von essenzieller Bedeutung sind. Die Kenntnis dieser Begriffe ist Voraussetzung für das Verständnis der behandelten Themen, da die Begriffe im weiteren Verlauf der Arbeit regelmäßig verwendet werden.

**App:** Der Begriff *App* stellt eine Abkürzung von *Applikation* dar und beschreibt Software, dessen Zweck in der Erfüllung der Bedürfnisse eines Benutzers besteht [7].

**Web-App:** Als *Web-App* werden Anwendungen bezeichnet, die unter Verwendung von Webtechnologien erstellt wurden. Zu den Webtechnologien zählen unter anderem *HTML* (Hypertext Markup Language), *CSS* (Cascading Style Sheets) und *JavaScript*, welche von Browsern genutzt werden. Das Hauptmerkmal von Web-Apps besteht in der Umsetzung von Benutzeroberflächen und Geschäftslogik [8].

**Progressive-Web-App:** Progressive-Web-Apps stellen eine Weiterentwicklung von Web-Apps dar, die sich durch eine erweiterte Funktionalität auszeichnen und auf die Ressourcen des Endgeräts zugreifen. Die Freigabe von JavaScript-Befehlen des Betriebssystems ermöglicht es Web-Apps, auf die Hardware des Geräts zuzugreifen. Die Nutzung der Funktionen erfolgt dabei in Abhängigkeit von den Einstellungen des Browsers [6].

**Asset-Management-System:** Ein *Asset-Management-System* (AMS) ist ein System zur Verwaltung von Vermögenswerten im Unternehmen oder in einer Organisation.

**Mobiles AMS:** Ein mobiles Asset-Management-System ermöglicht die Verwaltung von mobilen Vermögenswerten eines Unternehmens oder einer Organisation.

**Cross-Platform-Framework:** Der Begriff *Cross-Platform-Framework* bezeichnet einen Ansatz zur Erstellung plattformunabhängiger Anwendungen. Der hier beschriebene Ansatz erlaubt die Entwicklung und Bereitstellung einer App mit einer gemeinsamen Codebasis für verschiedene Betriebssysteme wie iOS, Android, Windows, macOS oder watchOS. Der Entwicklungsaufwand wird folglich reduziert, da die gleiche Codebasis für mehrere Plattformen genutzt werden kann. Dies führt zu einer effizienteren Entwicklung und Wartung der Anwendung.

**SQL:** SQL-Datenbanken basieren auf dem relationalen Datenbankmodell, bei dem Daten in tabellarischer Form organisiert sind. Jede Tabelle besteht aus Zeilen und

Spalten, wobei die Spalten aus fest definierten Datentypen bestehen. SQL-Datenbanken verwenden die Structured Query Language (SQL) für das Erstellen, Abrufen, Aktualisieren und Löschen von Daten.

**NoSQL:** *Not only SQL* (NoSQL) ist wie SQL ebenfalls eine Abfragesprache für Datenbanken. NoSQL-Datenbanken verzichten auf das starre relationale Modell und bieten eine flexible Datenstruktur. Sie unterstützen verschiedene Datenmodelle wie Dokumente, Schlüssel-Wert-Paare, Spalten oder graphische Modelle. NoSQL-Datenbanken sind darauf ausgelegt, unstrukturierte oder semi-strukturierte Daten zu speichern, die sich häufig ändern oder schwer in ein starres Schema passen.

**DBMS:** Ein *Datenbankmanagementsystem* (DBMS) dient der Steuerung, Verwaltung und Kontrolle von Daten innerhalb einer Datenbank. Zugriffe, wie z.B. das Anlegen, Lesen, Ändern oder Löschen von Datensätzen, des Anwenders oder der Anwendungssoftware werden über eine Kommunikationsschnittstelle ausgeführt.

**JSON:** *JavaScript Object Notation* (JSON) ist ein standardisiertes textbasiertes Format zur Darstellung strukturierter Daten auf der Grundlage der JavaScript-Objektsyntax. Es wird häufig für die Übermittlung von Daten in Webanwendungen verwendet. Ein JSON-Dokument ist eine Sammlung von Feldern und Werten im JSON-Format [9].

**Server:** Unter einem Server ist ein System innerhalb eines Netzwerks zu verstehen, das Informationen für andere Teilnehmer bereitstellt.

**Client:** Unter einem Client ist eine Instanz zu verstehen, die auf eine Informationsquelle innerhalb eines Netzwerks zugreifen möchte. Diesbezüglich ist zu erwähnen, dass es sich sowohl um eine natürliche Person als auch um ein System handeln kann.

**Ressource:** Der Begriff *Ressource* bezeichnet Informationen, welche verschiedenen Anwendungen zur Verfügung gestellt werden. Beispiele für Ressourcen sind Bilder, Videos, Texte oder sonstige Datensätze.

**API:** Die Bezeichnung *Application Programming Interface* (API) bezeichnet eine Programmierschnittstelle und definiert Regeln, um eine Kommunikation mit anderen Software-Systemen zu ermöglichen.

**HTTP:** Die Bezeichnung *Hypertext Transfer Protocol* (HTTP) ist ein Protokoll zum Datentransport für Webanwendungen [10].

**Local Storage:** Der *Local Storage* ist eine Methode zur Speicherung von Daten innerhalb eines Browsers. Sie ist Bestandteil der HTML5-Spezifikation [11]. Der lokale Speicher ermöglicht die Speicherung von Daten bis zu einer Größe von 10 Megabyte (MB), ohne dabei die Leistungsfähigkeit des Browsers zu beeinträchtigen.

**Authentifizierungs-Token:** Das Authentifizierungs-Token ist eine kryptische Zeichenfolge, die vom Server als Antwort auf eine Login-Anfrage generiert wird.

## 3 Stand der Technik

Die Erfassung des aktuellen Stands der Technik ist von Bedeutung, um einen Vergleich innovativer Technologien zu erörtern und Möglichkeiten für eine Umsetzung zu analysieren. Im Folgenden werden relevante Technologien zur Implementierung des AMS zusammengefasst und hinsichtlich der Vor- und Nachteile verglichen. Abschließend wird erörtert welche Technologie sich gut für eine Umsetzung dieser Arbeit eignet.

### 3.1 Mobile Datenerfassung

Technologien zur mobilen Datenerfassung lassen sich in zwei Hauptkategorien unterteilen: aktive und passive Datenträger. Aktive Datenträger wie ZigBee, Bluetooth und Wifi [12] ermöglichen den Datenaustausch über einen Microchip, bieten schnelle Verfügbarkeit und eine permanente Verbindung, erfordern jedoch eine zusätzliche Energieversorgung. Passive Datenträger wie Barcodes, QR-Codes und RFID [12] speichern weniger Informationen und müssen bei jedem Informationsabruf gescannt werden, benötigen dafür aber keine zusätzliche Energieversorgung.

Für diese Arbeit sind insbesondere passive Datenträger für die mobile Datenerfassung relevant. Die hohe Informationsdichte aktiver Datenträger ist für das AMS ist nicht notwendig und eine zusätzliche Energieversorgung gestaltet sich insbesondere bei Kunstwerken mit mehreren kleinen Komponenten als unpraktisch.

Unter den passiven Datenträgern stellen sich insbesondere QR-Codes als sehr praktikabel hervor. Das Scannen des Codes ist gegenüber Barcodes deutlich schneller und funktioniert auch unter ungünstigeren Bedingungen, wie Lichteinfall oder Winkel des Scanners. RFID stellt zwar auch eine gute Lösung dar, ist jedoch deutlichressourcenintensiver in der Beschaffung und zeitintensiver während der Entwicklung.

## 3.2 App Entwicklung

Die Entwicklung von Software für mobile Geräte hat sich in den letzten Jahrzehnten stark gewandelt. Während kurz nach Einführung des ersten Smartphones Apps gewöhnlicherweise direkt vom Betriebssystem, bspw. iOS oder Android, abhängig waren, werden heutzutage vermehrt sog. hybride Apps oder Web-Apps verbreitet. Nachfolgend werden Apps in drei Kategorien unterteilt: *native Apps*, *hybride Apps* und *Web-Apps* [5, 6].

Native Apps werden direkt unter Zuhilfenahme von plattformspezifischen Softwarebibliotheken entwickelt und werden deshalb auch als plattformspezifische App bezeichnet. Die Softwarebibliotheken sind hierbei so spezifisch, dass eine Software die bspw. für Android entwickelt wurde, nicht genau in der Form auch unter iOS eingesetzt werden kann. Zum einen verwendet Android *Java* bzw. *Kotlin* als grundlegende Programmiersprache, während unter iOS die Programmiersprache *Objective-C* bzw. *Swift* zum Einsatz kommt. Zum anderen gestaltet sich die Verwendung von Softwarebibliotheken, bspw. zur Ansteuerung der Kamera des Smartphones, für beide Betriebssysteme gänzlich unterschiedlich, da die Betriebssysteme inhärent unterschiedlich aufgebaut sind und unterschiedlichen Programmierparadigmen unterliegen [5, 6].

Darüber hinaus folgen die Benutzerführung und das Design auf unterschiedlichen Betriebssystemen auch plattformspezifischen Richtlinien. Während unter Android die Richtlinie *Material Design* sehr verbreitet ist, so ist unter iOS die Richtlinie *Cupertino Design* gängig und empfohlen [13, 14, 15]. Die Einhaltung dieser Richtlinien ist für Entwickler zwar nicht verpflichtend, jedoch steigern die Richtlinien die Qualität der App und bieten Benutzern eine intuitive, nahezu genormte, Möglichkeit die App zu bedienen. Bereits gelerntes Verhalten zur Bedienung einer App kann so in anderen Apps auch weiterverwendet werden [5, 6].

Hybride Apps bedienen sich einer Abstraktionsschicht und können so Programmcode auf mehrere Betriebssysteme abbilden. Daher werden sie auch als plattformübergreifende Apps bezeichnet. Die erwähnte Abstraktionsschicht wird als Cross Platform Framework bezeichnet und bietet einen einheitlichen Rahmen für den Aufruf von Funktionen, wie bspw. zur Ansteuerung der Kamera. So kann der gleiche Programmcode für mehrere Betriebssysteme verwendet werden. Programmcode hybrider Apps wird zum Teil oder gänzlich in nativen Code übersetzt. Das Cross Platform Framework ist

also vergleichbar mit einem Übersetzer für die verschiedenen Programmiersprachen und Betriebssysteme [5, 6].

Web-Apps sind über einen Browser, wie bspw. Firefox, Chrome oder Safari, vom Betriebssystem entkoppelt. Ähnlich wie bei hybriden Apps sind Web-Apps somit plattformübergreifend, basieren jedoch auf Web-Technologien. Sie werden also entweder unter Zuhilfenahme von üblichen Web-Technologien wie HTML, Java Script und CSS entwickelt oder nutzen ebenfalls ein Cross Platform Framework [5, 6].

Die Vorteile von nativen Apps liegen insbesondere in der Performance und in der Hardwareunterstützung. Native Apps erzielen eine sehr hohe Performance, da der komplizierte Code direkt mit dem Betriebssystem kommuniziert. In hybriden Apps kann komplizierter Code zu Schwierigkeiten führen, wenn besonders rechenintensive Operationen durchgeführt werden. Web-Apps zeigen die geringste Performance, da der Programmcode über den Browser an das Betriebssystem weitergeleitet werden muss und das Betriebssystem für den Browser nur begrenzte Rechenkapazität freigibt [16, 17].

Bei der Ansteuerung von Hardware ist das Betriebssystem maßgeblich beteiligt, da es detaillierte Informationen über eine sog. Hardwareabstraktionsschicht bereitstellt. Native Apps liegen also alle notwendigen Informationen zur Ansteuerung von Hardware vor. Für hybride Apps sind diese Informationen auch verfügbar, jedoch aus Sicherheitsgründen in einem geringeren Grad an Details. Noch geringer fällt der Grad an Details dann für Web-Apps aus [5, 18].

Das folgende Beispiel verdeutlicht diesen Zusammenhang. Während eine native App über sehr detaillierte Informationen der Kamera wie bspw. Hersteller, Fokuslänge, Auflösung, verwendete Objektive und Reaktionszeit verfügt, können hybride Apps und Web-Apps nur auf weniger Informationen zugreifen. Hybride Apps und Web-Apps ist es zwar dennoch möglich die Kamera anzusteuern, jedoch kann der plattformübergreifende Programmcode nicht verwendet werden, um die Kamera für einen spezifischen Fall sehr fein zu justieren.

Hybride Apps und Web-Apps weisen insbesondere Vorteile in der Entwicklungszeit auf, da eine Codebasis für mehrere Plattformen bzw. Betriebssysteme verwendet werden kann. Aus gleichem Grund sind hybride Apps und Web-Apps deutlich effizienter in der Wartung und in der Portierung auf weitere Betriebssysteme [5, 6].

Web-Apps weisen gegenüber hybriden und nativen Apps einen weiteren Vorteil in der Distribution der App auf. Eine Web-App muss von Benutzern nicht installiert werden und ist einfach über eine Adresse im Browser aufrufbar [5, 6].

Die folgende Tabelle stellt die genannten Kriterien zusammenfassend dar. Ein Pluspunkt markiert die Vorteile, ein Minuspunkt die Nachteile und ein Kreis weder Vor- noch Nachteil.

Kriterium	Native Apps	Hybride Apps	Web-Apps
Performance	+	o	-
Hardwareunterstützung	+	o	-
Entwicklungszeit	-	+	+
Wartbarkeit	-	+	+
Portierbarkeit	-	+	+
Distribution	o	o	+

Tab. 1: Vergleich von App-Kategorien

Insgesamt stellen hybride Apps und Web-Apps eine gute Lösung für dieses Projekt dar. Insbesondere die kurze Entwicklungszeit und die Wartbarkeit sind für die Implementierung von hoher Bedeutung.

Ein besonderes Kriterium in diesem Projekt betrifft die Distribution der App. Die App soll sowohl auf Desktop-PCs als auch auf mobilen Geräten installiert werden können. Außerdem soll die App nicht öffentlich zugänglich aus einem App-Store installiert werden, sondern über einen lokalen Server abrufbar sein. Um die Distribution zu vereinfachen, eignet sich eine Web-App zur Realisierung für dieses Projekt am besten.

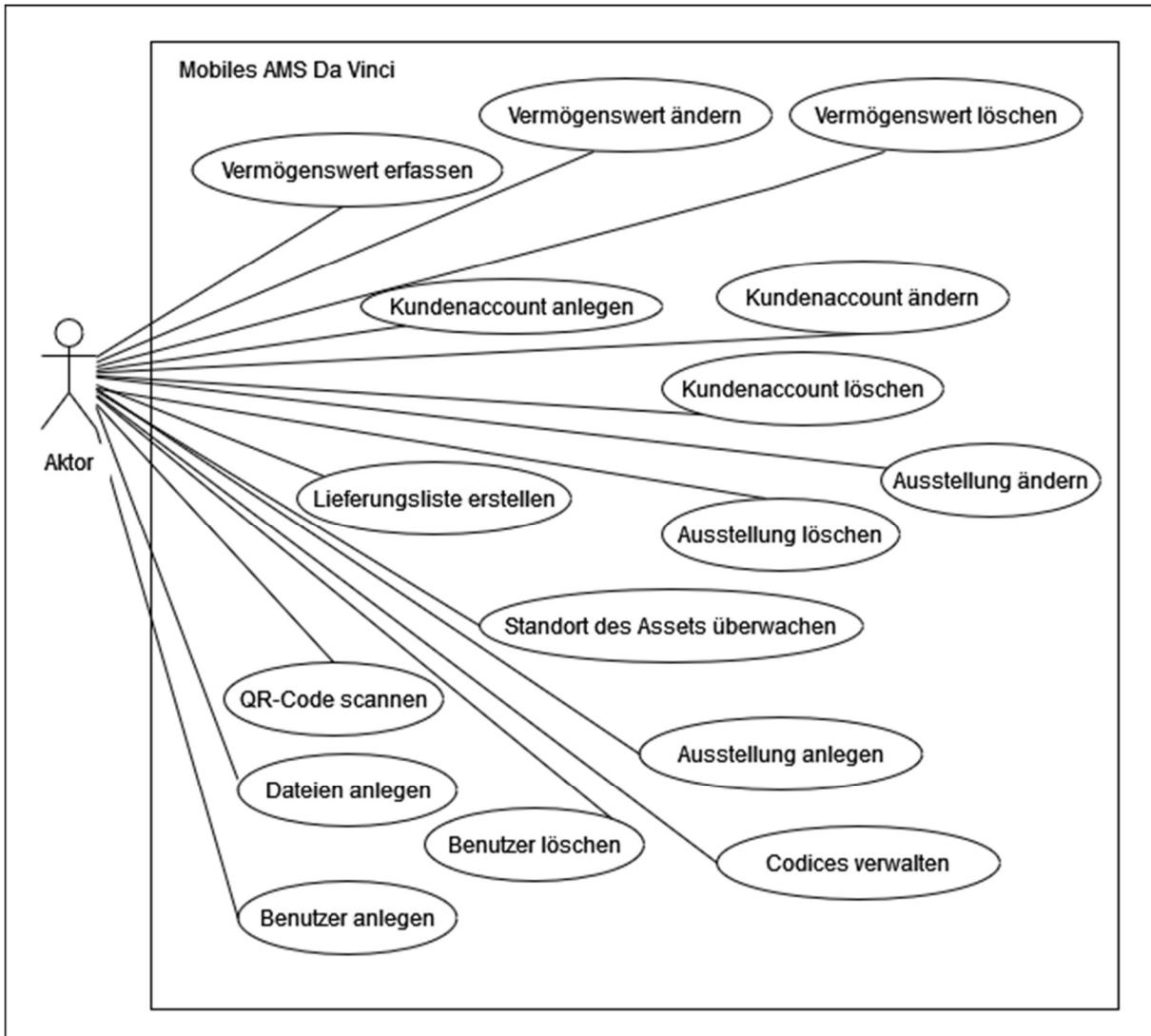
## 4 Anforderungen

In diesem Kapitel werden die Anforderungen an das System dargestellt. Unter Anforderungen sind die Erfassung von Funktionen, Ziele und Einschränkungen der Anwendung zu verstehen. Für die Erfassung der Anforderungen wurden semistruktuelle Interviews mithilfe von Fragebögen durchgeführt. Die Ergebnisse der Interviews werden im Anwendungsfalldiagramm und im Lastenheft dargestellt. Die Umsetzungskriterien

des Systems werden im Pflichtenheft abgebildet. Sowohl das Lasten- als auch das Pflichtenheft werden nach Anforderungsschablonen von Balzert erfasst [19].

## 4.1 Anwendungsfalldiagramm

Die gesammelten Daten aus den Interviews wurden als Grundlage für die Erstellung des Anwendungsfalldiagramms in Abb. 1 herangezogen. Abb. 1 präsentiert die Funkti-



onen des Systems aus der Perspektive der Benutzer. Das Anwendungsfalldiagramm dient als zentrale Referenz für die spätere Entwicklung des Systems und bildet die Grundlage für Erweiterung oder Einschränkung des Systemverhaltens in weiteren Phasen der Entwicklung.

Abb. 1: Anwendungsfalldiagramm des mobilen Asset-Management-Systems im Da Vinci-Projekt

## 4.2 Lastenheft

Das Lastenheft dient als Dokumentation für die Entwicklung eines mobilen Asset-Management-Systems (AMS). Es beschreibt die Zielsetzung, den geplanten Einsatz, die Produktfunktionen, die zu speichernden Daten, die Leistungsanforderungen sowie die Qualitätsanforderungen der zu entwickelnden Software.

### 4.2.1 Zielbestimmung

Die Software dient dem Personal mit Unterstützung bei der digitalen Verwaltung und Standortüberwachung von Vermögenswerten im Projekt Da Vinci. Des Weiteren soll das System die Kundenbetreuung und die Projektmitglieder während des Lieferungsprozesses unterstützen. Für die Standortüberwachung der Vermögenswerte und zur Identifikation von Assets im Rahmen des Lieferungsprozesses soll der GS1 Digital Link Standard eingesetzt werden [4]. Durch die Verknüpfung der Vermögenswerte mit GS1 Digital Link wird ein QR-Code erstellt, dessen Erkennung durch das System erfolgen soll.

### 4.2.2 Produkteinsatz

Das Produkt wird im Rahmen des Projekts Da Vinci der Hochschule Bielefeld angewendet. Die Zielgruppe umfasst alle Projektmitglieder.

### 4.2.3 Produktfunktionen

/PF10/ Erfassung, Änderung und Löschung von mechanischen Modellen.

/PF20/ Erweiterung und Löschung der Dokumente eines mechanischen Modells.

/PF30/ Erfassung, Änderung und Löschung vom Sockel eines mech. Modells.

/PF40/ Erfassung, Änderung und Löschung der Verpackung eines mech. Modells.

/PF50/ Erfassung, Änderung und Löschung von virtuellen Modellen.

/PF60/ Erfassung, Änderung und Löschung vom Banner.

/PF70/ Erfassung, Änderung und Löschung von Gestaltungselementen.

/PF80/ Erfassung, Änderung und Löschung von Skizzen.

/PF90/ Erfassung, Änderung und Löschung von Gemälden.

/PF100/ Erfassung, Änderung und Löschung von sonstigen Objekten.

/PF110/ Verknüpfung GS1 Digital Link mit den Assets.

/PF120/ Erkennung des Assets anhand des QR-Code.

/PF130/ Erfassung und Änderung des Standorts der Assets über einen QR-Code.

- /PF140/ Erfassung, Änderung und Löschung von Kunden.
- /PF150/ Erweiterung und Löschung der Dokumente eines Kunden.
- /PF160/ Erfassung und Änderung des Angebots eines Kunden.
- /PF170/ Erfassung, Änderung und Löschung von Ausstellungen.
- /PF180/ Erfassung und Änderung der Lieferung einer Ausstellung.
- /PF190/ Erweiterung und Löschung der Dokumente einer Ausstellung.
- /PF200/ Erstellung einer PDF-Datei für das Angebots des Kunden.
- /PF220/ Erstellung einer PDF-Datei für die Lieferung einer Ausstellung.
- /PF230/ Erfassung, Änderung und Löschung des Codex von Leonardo da Vinci.

#### 4.2.4 Produktdaten

/PD10/ Mechanisches Modell: Bezeichnung, Kürzung, Abmessungsparameter, Versicherung, Position, Pfad zum Bild, Begin der Ausleihe, Ende der Ausleihe, Beschreibung, Modelfamilie, Gewicht, Wasser, Strom, Bedienpersonal, Aufsteller, Betätigung, Bedienelement, Modellkarte, Sockel, Transport, Verpackung.

/PD20/ Virtuelles Modell: Bezeichnung, Kürzung, Kategorie, Abmessungsparameter, Pfad zum Bild, Beschreibung, Modelfamilie Aufsteller, Texturiert, Animiert, im Ausstellungsraum.

/PD30/ Banner, Gestaltungselemente, Skizzen, Gemälde und sonstigen Objekte: Bezeichnung, Kürzung, Kategorie, Abmessungsparameter, Versicherung, Position, Begin der Ausleihe, Ende der Ausleihe, Fixierung, Elementverpackung.

/PD40/ Kundenkundendaten: Name, Adresse, Kontaktdaten, Begin des Angebots, Ende des Angebots, Angebotsliste, Notizen.

/PD50/ Ausstellungsdaten: Bezeichnung, Adresse, Ausstellungseröffnung, Ausstellungsende, Aufbaudatum, Abbaudatum, Liste der Exponate, Notizen, Lieferung.

/PD60/ Lieferung: Vorbereitungsliste, Lieferliste, Datum der Lieferung.

/PD70/ Codex: Bezeichnung, Anzahl der Seiten, Seitenbezeichnung, Schlüsselwörter, Notizen.

#### 4.2.5 Produktleistungen

/PL10/ Bei der Listenausgabe der Funktionen /PF50/, /PF60/, /PF70/, /PF80/, /PF90/, /PF100/ werden zunächst die ersten 25 Elemente ausgegeben, weitere Elemente nur auf Wunsch.

## 4.2.6 Qualitätsanforderungen

Produktqualität	sehr gut	gut	normal	irrelevant
Funktionalität		x		
Zuverlässigkeit		x		
Benutzbarkeit	x			
Effizienz			x	
Änderbarkeit		x		
Übertragbarkeit				x

## 4.3 Pflichtenheft

Das Pflichtenheft baut auf den im Lastenheft definierten Anforderungen auf und dient als detaillierter Leitfaden für die Umsetzung des mobilen Asset-Management-Systems. Es legt die Muss-Anforderungen für das zu realisierende System fest und beschreibt auch Wunsch-Anforderungen, die optional umgesetzt werden können, falls es die zeitlichen Ressourcen des Entwicklers erlauben. Zudem stellt das Pflichtenheft die Abgrenzungskriterien klar dar, indem es definiert, welche Anforderungen und Funktionen nicht umgesetzt werden. Diese Abgrenzung hilft, den Umfang des Projekts präzise zu definieren und Missverständnisse bezüglich der Erwartungen und des Projektumfangs zu vermeiden. Im Folgenden wird das Pflichtenheft nach den Vorgaben von Balzert dargestellt [19].

### 4.3.1 Zielbestimmung

Das Ziel des Projekts ist die Entwicklung eines mobilen Asset-Management-Systems, das die digitale Verwaltung der Vermögenswerte vom Projekt *Da Vinci* ermöglicht. Die folgenden Kriterien bestimmen die Anforderungen an das System.

#### Musskriterien:

- Der Zugriff auf das System muss nur für autorisierte Benutzer stattfinden.
- Das System muss die Verwaltung eines Benutzers ermöglichen.
- Das System muss den Benutzer nach seiner Rolle unterscheiden.

- Das System muss die Verwaltung von mechanischen Modellen, virtuellen Modellen, Bannern, Gestaltungselementen, Skizzen, Gemälden und Gestaltungobjekten ermöglichen.
- Das System muss die Verwaltung des Kunden ermöglichen.
- Das System muss die Verwaltung von Ausstellungen ermöglichen.
- Eine flexible Datenbankstruktur muss vorhanden sein.
- Das System muss das Auslesen von PDF-Dateien gestatten.

**Wunschkriterien:**

- Jedem physischen Asset kann ein eindeutiger QR-Code zugewiesen sein.
- Das Scannen eines QR-Codes soll detaillierte Informationen über das Asset darstellen.
- Die Standortdaten der Benutzer können vom System ausgelesen werden und in der Datenbank gespeichert werden.
- Das System kann eine Standortüberwachung der Assets anbieten.
- Das System kann den Lieferungsprozess starten und beenden
- Das System kann die Codices von Leonardo da Vinci verwalten.

**Abgrenzungskriterien:**

- Das System wird nicht mit dem GS1 Digital Link Standard verknüpft, somit wird der dazugehörige QR-Code vom GS1 Digital Link nicht erstellt.
- Die Erkennung der Assets anhand des QR-Codes während des Lieferungsprozesses wird nicht umgesetzt.
- Es werden keine PDF-Dateien vom System erstellt.

### 4.3.2 Produkteinsatz

Das Produkt wird im Rahmen des Projekt *Da Vinci* der Hochschule Bielefeld angewendet. Die Zielgruppe umfasst alle Projektmitglieder.

### 4.3.3 Produktübersicht

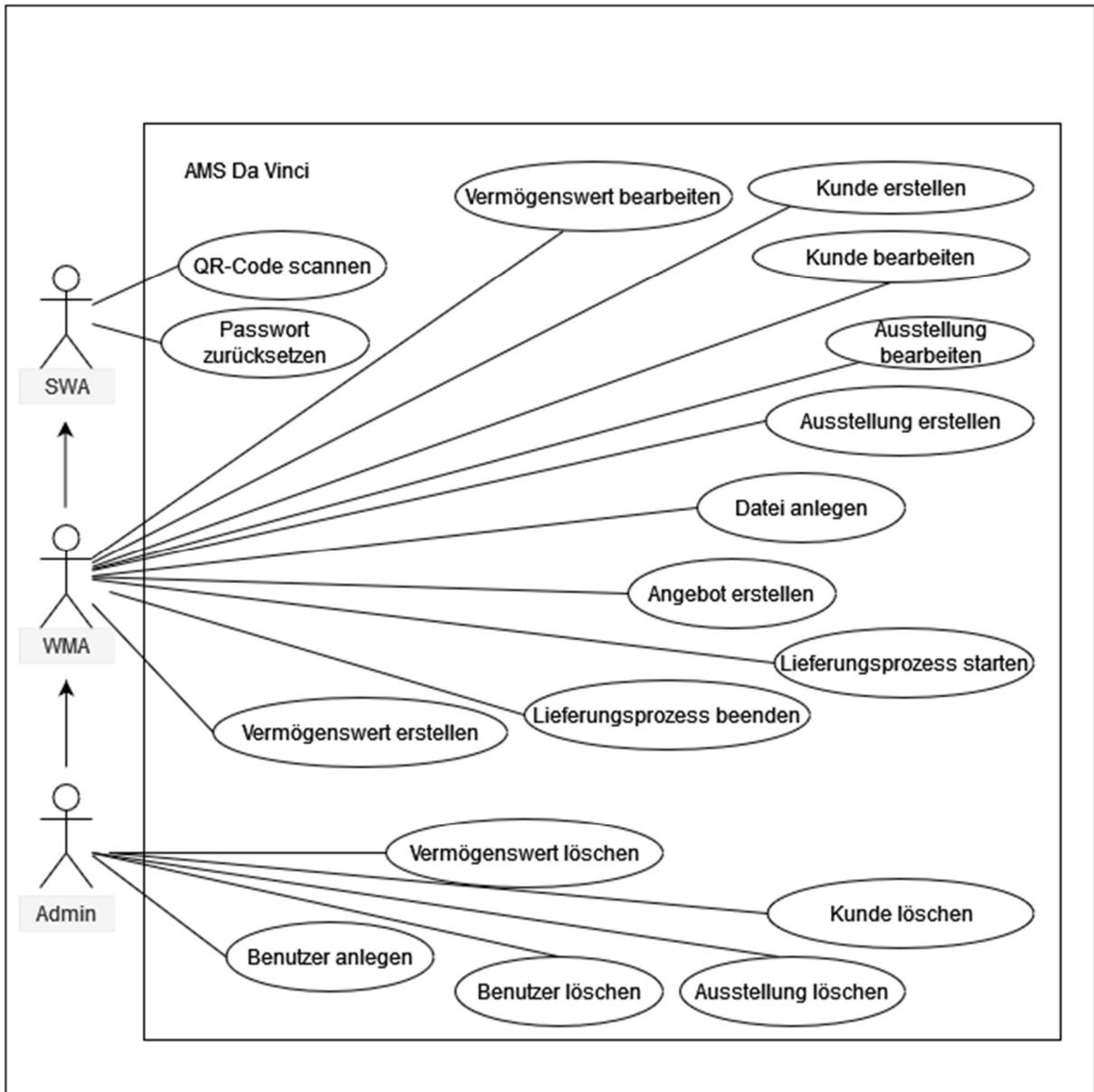


Abb. 2: Anwendungsfalldiagramm des Asset-Management-Systems im Da Vinci-Projekt. Admin steht für Administrator. WMA bedeutet wissenschaftliche Hilfskraft und SWA studentische Hilfskraft.

### 4.3.4 Produktfunktionen

Zu den Vermögenswerten sind mechanische Modelle, virtuellen Modelle, Banner, Gestaltungelemente, Skizzen, Gemälde und sonstige Gestaltungsobjekte zu verstehen.

/PF10/

Geschäftsprozess	Anmeldung im System.
Bedingungen	Benutzer ist im System angelegt.

Akteure	Admin, WMA, SMA, System
Ablauf	Benutzer füllt das Formular mit E-Mail und Passwort aus. System überprüft die Daten und verifiziert die Zugriffsrechte des Benutzers.
Auswirkung	Benutzer wird im System angemeldet.

/PF20/

Geschäftsprozess	Benutzer anlegen.
Bedingungen	Benutzer muss Admin Rechte besitzen.
Akteure	Admin, System
Ablauf	Admin wählt die Option <i>Neuen Benutzer anlegen</i> . System öffnet das Formular zur Eingabe von E-Mail, Passwort und Rolle des Benutzers. System speichert die Daten in der Datenbank und bestätigt die Erstellung.
Auswirkung	Neuer Benutzer wird in der Datenbank angelegt.

/PF30/

Geschäftsprozess	Password ändern.
Bedingungen	Benutzer muss authentifiziert sein.
Akteure	Admin, WMA, SMA, System
Ablauf	Benutzer wählt die Option <i>Passwort zurücksetzen</i> . System öffnet das Formular zur Eingabe des neuen Passwortes. System speichert die Daten in der Datenbank und bestätigt die Änderung.
Auswirkung	Benutzer kann beliebig oft sein Passwort im System ändern.

/PF40/

Geschäftsprozess	Benutzer löschen.
Bedingungen	Benutzer muss als Admin authentifiziert sein.
Akteure	Admin, System

Ablauf	Admin wählt die Option <i>Benutzer löschen</i> . System zeigt ein Formular mit der Benutzerliste. Admin wählt einen Benutzer zum Löschen aus. System fordert eine Bestätigung an. Nach Bestätigung löscht das System den Benutzer aus der Datenbank.
Auswirkung	Benutzer wird vom System entfernt und hat keinen Zugriff auf das System.

/PF50/

Geschäftsprozess	Vermögenswert erstellen.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.
Akteure	Admin, WMA, System
Ablauf	Benutzer wählt die Option <i>Neuen Vermögenswert erstellen</i> . System öffnet das Formular zur Eingabe des Vermögenswerts. Benutzer füllt die erforderlichen Felder aus und speichert den Vermögenswert. System generiert einen QR-Code für das Modell. System speichert die Daten in der Datenbank und bestätigt die Erstellung.
Auswirkung	Neuer Vermögenswert wird in der Datenbank angelegt.

/PF60/

Geschäftsprozess	Vermögenswert bearbeiten.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.
Akteure	Admin, WMA, System.
Ablauf	Benutzer wählt einen vorhandenen Vermögenswert zur Bearbeitung aus. System ruft die aktuellen Daten des Vermögenswerts aus der Datenbank ab und stellt sie in einem Bearbeitungsformular dar. Benutzer nimmt die gewünschten Änderungen vor. System speichert die aktualisierten Daten in der Datenbank und bestätigt die erfolgreiche Bearbeitung.

Auswirkung	Änderungen am Vermögenswert werden in der Datenbank aktualisiert.
------------	-------------------------------------------------------------------

/PF70/

Geschäftsprozess	Vermögenswert löschen.
Bedingungen	Benutzer muss als Admin authentifiziert sein.
Akteure	Admin, System
Ablauf	Admin wählt einen Vermögenswert zum Löschen aus. System fordert eine Bestätigung an. Nach Bestätigung löscht das System den Vermögenswert aus der Datenbank.
Auswirkung	Vermögenswert wird aus der Datenbank entfernt.

/PF80/

Geschäftsprozess	Mechanisches Modell um PDF-Datei erweitern.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.
Akteure	Admin, WMA, System
Ablauf	Benutzer wählt die Option <i>Dokument hinzufügen</i> . System zeigt eine Benutzeroberfläche zur Auswahl von Dateien an. Benutzer wählt eine PDF-Datei von seinem Gerät aus. Datei wird dem ausgewählten mechanischen Modell zugeordnet. System empfängt die Datei und speichert sie in der Datenbank.
Auswirkung	Mechanisches Modell wird um eine PDF-Datei erweitert. Datei kann angezeigt werden.

/PF90/

Geschäftsprozess	Mechanisches Modell um Sockel erweitern.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.
Akteure	Admin, WMA, System
Ablauf	Benutzer wählt die Option <i>Sockel hinzufügen</i> . System öffnet das Formular zur Eingabe der Sockel. Benutzer füllt die

	erforderlichen Felder aus und speichert den Sockel. System generiert einen QR-Code für den Sockel. System speichert die Daten in der Datenbank und bestätigt die Erstellung.
Auswirkung	Mechanisches Modell wird um Sockeldaten erweitert.

/PF100/

Geschäftsprozess	Mechanisches Modell um Zubehör erweitern.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.
Akteure	Admin, WMA, System
Ablauf	Benutzer wählt die Option <i>Zubehör hinzufügen</i> . System öffnet das Formular zur Eingabe des Zubehörs. Benutzer füllt die erforderlichen Felder aus. System fragt nach weiterem Zubehör. Benutzer bestätigt die Eingabe. System generiert einen QR-Code für Zubehör. System speichert die Daten in der Datenbank und bestätigt die Erstellung.
Auswirkung	Mechanisches Modell wird um Zubehör erweitert.

/PF110/

Geschäftsprozess	Mechanisches Modell um Verpackung für das Modell erweitern.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.
Akteure	Admin, WMA, System
Ablauf	Benutzer wählt die Option <i>Verpackung hinzufügen</i> . System öffnet das Formular zur Eingabe der Verpackung. Benutzer füllt die erforderlichen Felder aus. System fragt nach weiterer Verpackung. Benutzer bestätigt die Eingabe. System generiert einen QR-Code für eine einzelne Verpackung. System speichert die Daten in der Datenbank und bestätigt die Erstellung.
Auswirkung	Mechanisches Modell wird um Verpackungsdaten erweitert.

/PF120/

Geschäftsprozess	Sockel bearbeiten.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.
Akteure	Admin, WMA, System
Ablauf	Benutzer wählt einen vorhandenen Sockel zur Bearbeitung aus. System ruft die aktuellen Daten des Sockels aus der Datenbank ab und stellt sie in einem Bearbeitungsformular dar. Benutzer nimmt die gewünschten Änderungen vor. System speichert die aktualisierten Daten in der Datenbank und bestätigt die erfolgreiche Bearbeitung.
Auswirkung	Mechanisches Modell wird um Sockeldaten erweitert.

/PF130/

Geschäftsprozess	Kunde erstellen.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.
Akteure	Admin, WMA, System
Ablauf	Benutzer wählt die Option <i>Neuen Kunde anlegen</i> . System öffnet das Formular zur Eingabe der Kundendaten. Benutzer füllt die erforderlichen Felder aus und speichert den Kunden. System speichert die Daten in der Datenbank und bestätigt die Erstellung.
Auswirkung	Neuer Kunde wird in der Datenbank angelegt.

/PF140/

Geschäftsprozess	Kunde bearbeiten
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein
Akteure	Admin, WMA, System
Ablauf	Benutzer wählt einen vorhandenen Kunden zur Bearbeitung aus. System ruft die aktuellen Daten des Kunden aus der Datenbank ab und stellt sie in einem Bearbeitungsformular dar. Benutzer nimmt die gewünschten Änderungen vor. Das

	System speichert die aktualisierten Daten in der Datenbank und bestätigt die erfolgreiche Bearbeitung.
Auswirkung	Kunde wird in der Datenbank aktualisiert.

/PF150/

Geschäftsprozess	Kunde löschen.
Bedingungen	Benutzer muss als Admin authentifiziert sein. Dem Kunden ist keine Ausstellung zugewiesen.
Akteure	Admin, System
Ablauf	Admin wählt einen Kunden zum Löschen aus. System fordert eine Bestätigung an. System prüft, ob dem Kunden keine Ausstellung zugewiesen ist. Nach Überprüfung löscht das System den Kunden aus der Datenbank.
Auswirkung	Kunde wird aus der Datenbank entfernt.

/PF160/

Geschäftsprozess	Kundendaten um PDF-Datei erweitern.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.
Akteure	Admin, WMA, System
Ablauf	Benutzer wählt die Option <i>Dokument hinzufügen</i> . System zeigt eine Benutzeroberfläche zur Auswahl von Dateien an. Benutzer wählt eine PDF-Datei von seinem Gerät aus. Datei wird dem ausgewählten Kunden zugeordnet. System empfängt die Datei und speichert sie in der Datenbank.
Auswirkung	Kunde wird um eine PDF-Datei erweitert. Datei kann angezeigt werden.

/PF170/

Geschäftsprozess	Angebot erstellen.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.

Akteure	Admin, WMA, System
Ablauf	<p>Benutzer wählt die Option <i>Angebot erstellen</i>. System öffnet das Formular zur Eingabe des Zeitraums des Angebots. Benutzer füllt die erforderlichen Felder aus und speichert den Zeitraum. System speichert die eingegebenen Daten in der Datenbank und zeigt den festgelegten Zeitraum beim entsprechenden Kunden an. Benutzer kann das Angebot um mechanische, virtuelle Modelle und Gestaltungselemente erweitern.</p> <p>Benutzer wählt die Option <i>Mechanische Modelle auswählen</i>. System öffnet das Formular mit der Modelliste. Benutzer wählt beliebig viele Modelle aus und speichert die Auswahl. System schließt das Formular. Benutzer wählt die Option <i>Virtuelle Modelle auswählen</i>. System öffnet das Formular mit der Modelliste. Benutzer wählt beliebig viele Modelle aus und speichert die Auswahl. System schließt das Formular. Benutzer wählt die Option <i>Gestaltungselemente auswählen</i>. System öffnet das Formular mit den Gestaltungselementen. Benutzer wählt beliebig viele Elemente aus und speichert die Auswahl. System schließt das Formular. Benutzer speichert das Angebot. System speichert das Angebot in der Datenbank und zeigt die Angebotsliste.</p>
Auswirkung	Zeitraum für das Angebot und die Angebotsliste wird in der Datenbank angelegt.

/PF180/

Geschäftsprozess	Zeitraum für das Angebot der Kunde bearbeiten.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.
Akteure	Admin, WMA, System
Ablauf	Benutzer wählt die Option <i>Zeitraum für das Angebot bearbeiten</i> . Das System ruft den Zeitraum aus der Datenbank ab und stellt es in einem Bearbeitungsformular dar. Benutzer korrigiert

	die erforderlichen Felder und speichert den Zeitraum. System speichert die Daten und zeigt die Änderungen an.
Auswirkung	Zeitraum für das Angebot der Kunde wird aktualisiert.

/PF190/

Geschäftsprozess	Angebot des Kunden für mechanische Modelle korrigieren.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.
Akteure	Admin, WMA, System
Ablauf	Benutzer wählt die Option <i>Mechanische Modelle zum Angebot hinzufügen</i> . System ruft das Angebot des Kunden mit der Liste von mechanischen Modelle aus der Datenbank ab. System zeigt eine Benutzeroberfläche zur Auswahl von Modellen an. Die bereits im Angebot enthaltenen mechanischen Modelle sind farblich markiert. Benutzer wählt zusätzliche mechanische Modelle oder entfernt bereits ausgewählte Modelle. System zeigt geänderte Markierungen an. Benutzer speichert die aktualisierte Angebotsliste. Das System speichert diese in der Datenbank.
Auswirkung	Angebot des Kunden wird in der Datenbank aktualisiert.

/PF200/

Geschäftsprozess	Angebot des Kunden für virtuelle Modelle korrigieren
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein
Akteure	Admin, WMA, System
Ablauf	Benutzer wählt die Option <i>Virtuelle Modelle zum Angebot hinzufügen</i> . System ruft das Angebot der Kunde mit der Liste der virtuellen Modelle aus der Datenbank ab. System zeigt eine Benutzeroberfläche zur Auswahl von Modellen an. Die bereits im Angebot enthaltenen virtuellen Modelle sind farblich markiert. Benutzer wählt zusätzliche virtuelle Modelle oder entfernt bereits ausgewählte Modelle. System zeigt geänderte

	Markierungen an. Benutzer speichert aktualisierte Angebotsliste. System speichert diese in der Datenbank.
Auswirkung	Angebot des Kunden wird in der Datenbank aktualisiert.

/PF210/

Geschäftsprozess	Angebot des Kunden für Banner und Gestaltungselemente korrigieren.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.
Akteure	Admin, WMA, System
Ablauf	Benutzer wählt die Option <i>Banner und Gestaltungselemente zum Angebot hinzufügen</i> . System ruft das Angebot des Kunden mit der Liste der Banner und Gestaltungselemente aus der Datenbank ab. System zeigt eine Benutzeroberfläche zur Auswahl von Bannern und Gestaltungselementen an. Die bereits im Angebot enthaltenen Banner und Gestaltungselemente sind farblich markiert. Benutzer wählt zusätzliche Banner und Gestaltungselemente oder entfernt bereits ausgewählte Elemente. System zeigt geänderte Markierungen an. Benutzer speichert die aktualisierte Angebotsliste. Das System speichert diese in der Datenbank.
Auswirkung	Angebot des Kunden wird in der Datenbank aktualisiert.

/PF220/

Geschäftsprozess	Ausstellung erstellen.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein. Ausstellung wird einem Kunden zugewiesen. Kunde muss Angebot besitzen.
Akteure	Admin, WMA, System
Ablauf	Benutzer wählt die Option <i>Neue Ausstellung anlegen</i> . System öffnet das Formular zur Eingabe der Ausstellung. Benutzer füllt die erforderlichen Felder aus und speichert die Ausstellung.

	Das System speichert die Daten in der Datenbank und bestätigt die Erstellung.
Auswirkung	Ausstellung wird in der Datenbank gespeichert.

/PF230/

Geschäftsprozess	Ausstellung bearbeiten.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.
Akteure	Admin, WMA, System
Ablauf	Benutzer wählt eine vorhandene Ausstellung zur Bearbeitung aus. System ruft die aktuellen Daten aus der Datenbank ab und stellt sie in einem Bearbeitungsformular dar. Benutzer nimmt die gewünschten Änderungen vor. System speichert die aktualisierten Daten in der Datenbank und bestätigt die erfolgreiche Bearbeitung.
Auswirkung	Ausstellung wird in der Datenbank aktualisiert.

/PF240/

Geschäftsprozess	Ausstellung löschen.
Bedingungen	Benutzer muss als Admin authentifiziert sein.
Akteure	Admin, System
Ablauf	Benutzer wählt eine Ausstellung zum Löschen aus. System fordert eine Bestätigung an. Nach Bestätigung löscht das System die Ausstellung aus der Datenbank.
Auswirkung	Ausstellung wird aus der Datenbank entfernt.

/PF250/

Geschäftsprozess	Ausstellung um PDF-Datei erweitern.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.
Akteure	Admin, WMA, System

Ablauf	Benutzer wählt die Option <i>Dokument hinzufügen</i> . System zeigt eine Benutzeroberfläche zur Auswahl von Dateien an. Benutzer wählt ein PDF-Dokument von seinem Gerät aus. System empfängt die Datei und speichert sie in der Datenbank. Die Datei wird dem ausgewählten Kunden zugeordnet.
Auswirkung	Ausstellung wird um eine PDF-Datei erweitert. Datei kann angezeigt werden.

/PFW260/

Geschäftsprozess	QR-Code scannen.
Bedingungen	Benutzer muss im System angemeldet sein
Akteure	Admin, WMA, SMA, System
Ablauf	Benutzer wählt die Option <i>Kamera öffnen</i> . Das System greift auf die Kamera des Benutzergeräts zu. Der Benutzer scannt den QR-Code. System öffnet die dem QR-Code zugewiesenen Daten.
Auswirkung	Benutzer kann QR-Code auslesen und Daten im System anzeigen.

/PFW270/

Geschäftsprozess	Lieferung starten.
Bedingungen	Benutzer muss als Admin oder WMA authentifiziert sein.
Akteure	Admin, WMA, SMA, System
Ablauf	Der Benutzer wählt die Option <i>Lieferung starten</i> . Das System öffnet die Benutzeroberfläche mit den Lieferungspositionen. Benutzer prüft Positionen und speichert eine Vorbereitungsliste im System.
Auswirkung	Lieferungsstartdatum und Vorbereitungsliste mit geprüften Positionen wird zur Ausstellung hinzugefügt.

#### 4.3.5 Produktdaten

/PD10/ Mechanisches Modell: Bezeichnung, Kürzung, Kategorie, Abmessungsparameter, Versicherung, Position, Pfad zum Bild, Pfad zum QR-Code, Begin der Ausleihe, Ende der Ausleihe, Beschreibung, Modelfamilie, Gewicht, Wasser, Strom, Bedienpersonal, Aufsteller, Betätigung, Bedienelement, Modellkarte, Sockel, Transport, Verpackung, Pfad zu Dokumenten.

/PD20/ Virtuelles Modell: Bezeichnung, Kürzung, Kategorie, Abmessungsparameter, Pfad zum Bild, Pfad zum QR-Code, Beschreibung, Modelfamilie, Aufsteller, Texturiert, Animiert, im Ausstellungsraum.

/PD30/ Banner, Gestaltungselemente, Skizzen, Gemälde und sonstige Objekte: Bezeichnung, Kürzung, Kategorie, Abmessungsparameter, Versicherung, Position, Pfad zum Bild, Pfad zum QR-Code, Begin der Ausleihe, Ende der Ausleihe, Fixierung, Elementverpackung.

/PD40/ Kundenkundendaten: Name, Adresse, Kontaktdaten, Anfang des Angebots, Ende des Angebots, Angebotsliste, Notizen, Pfad zu Dokumenten.

/PD50/ Ausstellungsdaten: Bezeichnung, Adresse, Ausstellungseröffnung, Ausstellungsende, Aufbaudatum, Abbaudatum, Liste der Exponate, Notizen, Pfad zu Dokumenten, Lieferung.

/PD60/ Lieferung: Vorbereitungsliste, Lieferliste, Datum der Lieferung.

/PD70/ Codex: Bezeichnung, Anzahl der Seiten, Seitenbezeichnung, Schlüsselwörter, Notizen.

/PD80/ Sockel: Bezeichnung, Abmessungsparameter, Standort, Pfad zum QR-Code, Stapelbar, Palette, Abmessung der Palette.

/PD90/ Zubehör: Bezeichnung, Anzahl, Standort, Pfad zum QR-Code.

/PD100/ Modellverpackung: Bezeichnung, Verpackungstyp, Abmessungsparameter, Standort, Pfad zum QR-Code, Stapelbar, Palette, Abmessung der Palette, Notiz.

#### 4.3.6 Produktleistungen

/PL10/

Bezeichnung	Allgemeine Systemreaktionszeit
-------------	--------------------------------

Anforderung	System sollte auf Benutzeraktionen und -anfragen in akzeptabler Zeit reagieren.
Antwortzeit für Benutzerinteraktionen	Idealweise unter 1 Sekunde.
Ladezeiten für Daten:	Idealweise unter 2 Sekunden für das Abrufen und Anzeigen von Daten.
Kommentar	Diese Zeiten bieten eine grundlegende Orientierung für eine akzeptable Benutzererfahrung, ohne spezifische Anforderungen zu erfüllen.

/PL20/

Bezeichnung	Zeit zur Ausführung der Datenbankoperationen
Anforderung	Zeit zur Ausführung von Datenbankoperationen wie das Speichern, Abrufen und Aktualisieren darf einen definierten Schwellenwert nicht überschreiten.
Details	Speichern: Maximal 500 ms. Abrufen: Maximal 300 ms. Aktualisieren: Maximal 500 ms.
Kommentar	Dient der Gewährleistung einer reaktionsschnellen Benutzeroberfläche.

/PL30/

Bezeichnung	Datentransfervolumen
Anforderung	Die Menge der übertragenen Daten und die Dauer des Datentransfers müssen innerhalb akzeptabler Grenzen liegen, um eine effiziente Kommunikation sicherzustellen.
Details	Datenvolumen pro Dokument: Maximal 5 MB (umfasst PDF-Dateien und Bilder)
Kommentar	Stellt sicher, dass auch bei größeren Datenmengen die Übertragungszeiten innerhalb akzeptabler Grenzen bleiben.

/PL40/

Bezeichnung	Listenausgabe für Skizzen, Banner, Gestaltungselemente, Gemälde und sonstige Gestaltungsobjekte
Anforderung	Bei der Listenausgabe der Funktion /PF50/ für Skizzen, Banner, Gestaltungselemente, Gemälde und Sonstige Gestaltungsobjekte werden zunächst die ersten 25 Elemente ausgegeben, weitere Elemente nur auf Wunsch.
Details	Benutzeroberfläche bei der Anzeige von umfangreichen Listen nicht überlastet wird und gewährleistet eine schnelle Antwortzeit.
Kommentar	Optimiert die Ladezeiten und verbessert die Benutzerfreundlichkeit, insbesondere bei großen Datenmengen.

#### 4.3.7 Qualitätsanforderungen

Produktqualität	sehr gut	gut	normal	irrelevant
Funktionalität		x		
Zuverlässigkeit		x		
Benutzbarkeit	x			
Effizienz			x	
Änderbarkeit		x		
Übertragbarkeit				x

#### 4.3.8 Benutzeroberfläche

/B01/

Layout	Hauptfenster dient als zentrale Benutzeroberfläche der Anwendung. Es umfasst eine Navigationsleiste, Kopfleiste, einen Inhaltsbereich und eine Statusleiste. Navigationsleiste befindet sich entweder am oberen Rand oder an der Seite des Fensters. Sie bietet Zugriff auf die Hauptmodule der Anwendung. Inhaltsbereich ist der zentrale Bereich, in dem
--------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	Formulare, Listen und Detailansichten angezeigt werden. Kopfleiste liegt am oberen Rand des Fensters und enthält zusätzliche Schaltflächen. Statusleiste liegt am unteren Rand des Fensters und zeigt Statusinformationen von Operationen oder Fehlerwarnungen. Statusleiste wird nur für einige Sekunden sichtbar.
--	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

/B02/

Dialogstruktur/Formulare	Dialoge beinhalten Formulare, die zur Eingabe und Bearbeitung von Daten vorgesehen sind. Sie enthalten erforderliche Eingabefelder, Validierungsregeln und Hilfetexte, um Benutzerfehler zu minimieren.
--------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

/B03/

Pop-up-Fenster:	Pop-up-Fenster sind für Bestätigungen des Löschvorgangs zu verwenden.
-----------------	-----------------------------------------------------------------------

/B04/

Zugriffrech SMA	SMA haben nur Lesezugriff auf das System und können /PF10/, /PF30/ ausführen.
-----------------	-------------------------------------------------------------------------------

/B05/

Zugriffrech WMA	WMA besitzen alle Zugriffrechte eines SMA und greifen auf Funktionen zur Erstellung und Bearbeitung von Vermögenswerten, Kunden, Angeboten und Ausstellungen zu. Sie können das System um PDF-Dateien erweitern.
-----------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

/B06/

Zugriffrech Admin	Ein Admin verfügt über uneingeschränkten Zugriff auf alle Funktionen und Konfigurationsmöglichkeiten der Anwendung. Dies umfasst das Anlegen, Bearbeiten und Löschen von Datensätzen.
-------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### 4.3.9 Nichtfunktionale Anforderungen

Das System muss eine hohe Sicherheit aufweisen, indem die Daten nur für autorisierte Benutzer zu sehen sind. Es muss auf gängigen Webbrowsern (Chrome, Firefox, Microsoft Edge, Safari) auf mobilen Geräten und Desktop-PCs lauffähig sein. Die Oberfläche des Systems soll benutzerfreundlich gestaltet sein, mit großen, gut lesbaren Texten. Es soll eine helle Farbpalette verwendet werden, um eine angenehme und zugängliche Benutzererfahrung zu gewährleisten.

### 4.3.10 Technische Produktumgebung

Die Web-Anwendung muss mit den folgenden Browsern kompatibel sein:

- Chrome: Version 96 und höher.
- Firefox: Version 99 und höher.
- Safari: Version 15.6 und höher.
- Microsoft Edge: Version 96 und höher.

Die in diesem Kapitel beschriebenen Anforderungen bilden die Grundlage für die gesamte Systementwicklung. Im nächsten Kapitel wird das Systemkonzept vorgestellt, das auf den Vorgaben des Pflichtenhefts aufbaut und die Systemarchitektur sowie die wesentlichen Komponenten erläutert.

## 5 Konzept

Das folgende Kapitel präsentiert das Konzept des zu entwickelnden Systems. Die Konzeptdarstellung erstreckt sich von der Perspektive der Entwickler bis zum Endverbraucher. Im ersten Schritt wird die zu entwickelnde Systemarchitektur präsentiert und im Anschluss erfolgt die Erläuterung einzelner Systemkomponenten.

### 5.1 Architektur

Im Rahmen der Anforderungen des Pflichtenhefts ist die Entwicklung eines Systems vorgesehen, welches sowohl die Speicherung von Daten als auch den Zugriff auf letztere für die Endnutzer ermöglicht. Ein weiteres wesentliches Kriterium stellt die Flexibilität und Skalierbarkeit des Systems dar.

Zur Realisierung der genannten Anforderungen wird eine Datenbank entwickelt, welche die Funktion des zentralen Elements zur Datenspeicherung übernimmt. Um den

Endnutzern den Zugriff auf die Datenbank zu ermöglichen, wird eine entsprechende Benutzeroberfläche erstellt. Die Benutzeroberfläche stellt somit das Instrument zur Interaktion mit der Datenbank sowie zum Zugriff auf die gespeicherten Informationen dar.

Die vorliegende Architektur basiert auf der Trennung von Benutzeroberfläche und Datenbank. Die Kommunikation zwischen beiden Komponenten erfolgt über eine neu entwickelte, vermittelnde Schicht. Diese fungiert als Schnittstelle, über welche die Benutzeroberfläche mit der Datenbank interagiert. Die nachfolgende Abbildung präsentiert eine grafische Darstellung der Architektur der Anwendung.

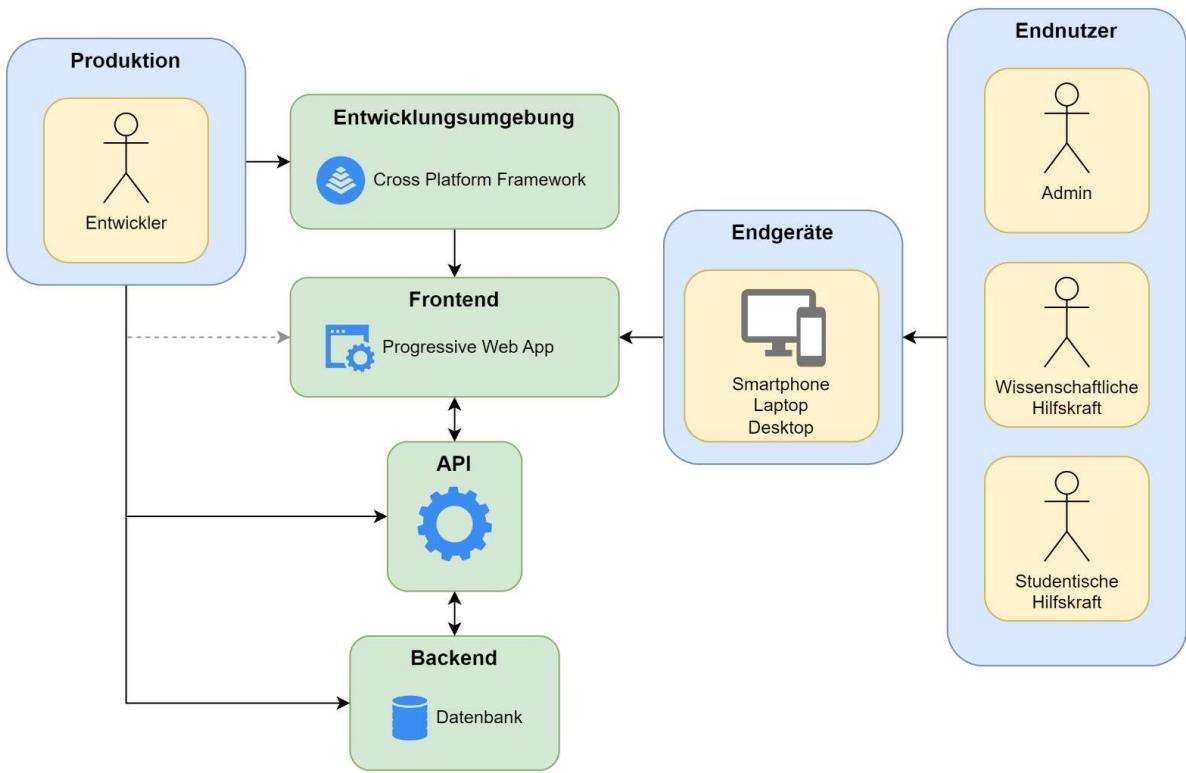


Abb. 3: Systemarchitektur.

Die Architektur der Anwendung umfasst folgende wesentliche Komponenten: Frontend, API und Backend. Die Abbildung verdeutlicht, dass sowohl Entwickler als auch Endnutzer auf das System zugreifen können. Die Entwicklung der Frontend-Schicht des Systems erfolgt durch die Nutzung eines Cross-Platform-Frameworks, wodurch eine plattformübergreifende Software-Entwicklung ermöglicht wird.

Die Realisierung des Frontends der Software erfolgt als Progressive-Web-App (PWA), welche in einem Webbrowser läuft und wie eine native App funktioniert. Die PWA ist für eine Vielzahl von Endgeräten optimiert, darunter Smartphones, Laptops und

Desktops. Die Benutzeroberfläche stellt die Verbindung zwischen dem System und den Endnutzern dar und ermöglicht die Interaktion mit dem System.

Das Backend umfasst die Serverlogik der Anwendung und ist für die Verarbeitung der Daten verantwortlich. Des Weiteren übernimmt es die Verwaltung der Geschäftslogik sowie den Betrieb einer Datenbank. Die Verbindung zwischen Backend und Frontend wird durch die API gewährleistet.

Die API fungiert als Vermittler zwischen dem Frontend und dem Backend. Die Kommunikation sowie der Datenaustausch zwischen den beiden Schichten werden durch die Schnittstelle ermöglicht, indem Anfragen des Frontends an das Backend weitergeleitet und entsprechende Antworten zurück an das Frontend übermittelt werden.

Die Endnutzer greifen über verschiedene Endgeräte auf das System zu. Dabei werden Benutzerrollen differenziert, beispielsweise Administratoren, wissenschaftliche Hilfskräfte und studentische Hilfskräfte. Diese verfügen jeweils über spezifische Zugriffsrechte und Aufgaben innerhalb des Systems.

Die vorliegende Struktur entspricht einer Client-Server-Architektur, wobei das Frontend als Client fungiert und Anfragen an das Backend (den Server) sendet, um Daten abzurufen oder zu verarbeiten. Die klare Trennung der Aufgaben zwischen Client und Server erlaubt eine unabhängige Entwicklung und den Betrieb der Benutzeroberfläche und der Datenverarbeitung und -speicherung. Im Folgenden werden die spezifischen Anforderungen und die Umsetzung des Frontends detailliert beschrieben.

## 5.2 Frontend

Das Frontend stellt die Benutzeroberfläche dar, über welche der Endbenutzer mit der Anwendung interagiert. Im Folgenden erfolgt die Definition der Anforderungen an das Frontend. Das Frontend umfasst sämtliche visuellen Elemente sowie Benutzerfunktionen. Es ist zu gewährleisten, dass das Frontend auf mobilen Geräten wie eine native Applikation funktioniert. Nach der Freigabe des Standorts soll eine Abfrage an den Benutzer über das Frontend erfolgen. Des Weiteren ist ein Zugriff der Anwendung auf die Webcam des Endgeräts erforderlich. Der Benutzer hat die Möglichkeit, die Kamera ein- und auszuschalten, um QR-Codes zu scannen.

Das Frontend soll mit einem System entwickelt werden, das plattformübergreifend funktionieren soll. Das System muss skalierbar sein, damit es zukünftig um

Applikationen für alle gängigen Betriebssysteme erweitert werden kann. Darüber hinaus soll es pflegeleicht sein, um die Aktualität des Systems sicherzustellen.

Um eine optimale Usability zu gewährleisten, ist es erforderlich, dass die Applikation eine Differenzierung zwischen mobilen, tabletbasierten und Desktop-Oberflächen vornimmt. Der Abruf und das Anlegen von Daten soll sowohl auf mobilen Geräten als auch auf dem Desktop möglich sein, wobei dem Anlegen von Daten auf dem Desktop eine höhere Relevanz beigemessen wird. Des Weiteren soll auf sämtlichen Geräten die Möglichkeit gegeben sein, neue Datensätze zu erstellen, zu bearbeiten und zu löschen. Um eine optimale Usability zu gewährleisten, ist es erforderlich, dass die Applikation eine Differenzierung zwischen mobilen, tabletbasierten und Desktop-Oberflächen vornimmt. Der Abruf und das Anlegen von Daten soll sowohl auf mobilen Geräten als auch auf dem Desktop möglich sein, wobei dem Anlegen von Daten auf dem Desktop eine höhere Relevanz beigemessen wird. Des Weiteren soll auf sämtlichen Geräten die Möglichkeit gegeben sein, neue Datensätze zu erstellen, zu bearbeiten und zu löschen.

Im folgenden Abschnitt werden die Anforderungen an API präsentiert, die als Vermittler zwischen Frontend und Backend die notwendige Funktionalität bereitstellt, um Datensätze effizient zu erstellen, zu bearbeiten und zu löschen.

### 5.3 API

Die API soll eine vollständige Trennung zwischen Frontend und Backend unterstützen, sodass eine klare Abgrenzung der Verantwortlichkeiten gewährleistet ist. Dies impliziert, dass Modifikationen in der Datenbank vorgenommen werden können, ohne dass eine Neuschreibung der Anwendungslogik erforderlich ist. Des Weiteren soll die API die Entwicklung des Frontends und des Backends in unterschiedlichen Programmiersprachen ermöglichen, um eine größtmögliche Flexibilität sicherzustellen. Die API sollte zudem sicher und effizient sein, um eine reibungslose und verlässliche Kommunikation zwischen den verschiedenen Komponenten der Anwendung zu gewährleisten.

Des Weiteren sollte die API so konzipiert werden, dass sie eine transparente, konsistente Schnittstelle bietet, die leicht nachvollziehbar ist. Dies erleichtert die Integration neuer Funktionen und die Wartung der Anwendung.

## 5.4 Backend

Das Backend umfasst die Entwicklung der Datenbankstruktur. Das Datenbanksystem muss in der Lage sein, große Datenmengen effizient zu verwalten, geringe Latenzzeiten aufzuweisen und flexible Datenmodelle zu unterstützen.

Es ist sicherzustellen, dass die Datenbank eine hohe Skalierbarkeit aufweist und eine hohe Verfügbarkeit der Daten gewährleistet werden kann. Des Weiteren ist zu berücksichtigen, dass eine Änderung der Datenmodelle in Zukunft nicht ausgeschlossen werden kann. Daher sollte das Backend so konzipiert sein, dass eine Adaption an zukünftige Erweiterungen und Modifikationen ohne signifikanten Aufwand gewährleistet ist. Im Anschluss ist im Backend die Integration von Sicherheits-mechanismen erforderlich, um den Zugriff auf die Daten zu kontrollieren und die Integrität sowie die Vertraulichkeit der Daten zu gewährleisten.

Das Konzept sieht die Realisierung der Anwendung mit einer Client-Server-Architektur vor, die eine klare Trennung zwischen Frontend und Backend ermöglicht. Im nächsten Kapitel wird die technische Umsetzung des Konzepts präsentiert.

# 6 Implementierung

Im vorliegenden Kapitel folgt eine detaillierte Beschreibung der technischen Realisierung einer PWA. Der Schwerpunkt liegt dabei auf der Implementierung der verschiedenen Systemkomponenten.

Zu Beginn werden technische Grundlagen sowie die Einrichtung der Entwicklungsumgebung erläutert. Anschließend wird die Implementierung des Frontends mittels Flutter im Detail dargelegt. Danach wird die Representational State Transfer API (REST-API), die mithilfe des Express.js-Frameworks in einer Node.js-Umgebung entwickelt wurde, behandelt. Abschließend wird die Datenbankstruktur vorgestellt, die unter Verwendung von MongoDB implementiert wurde.

## 6.1 Technische Grundlagen

In diesem Kapitel werden die technischen Grundlagen der Web-Anwendung aus der Perspektive der Entwickler sowie der Endbenutzer erläutert. Des Weiteren werden die erforderlichen Systemvoraussetzungen definiert und die Einrichtung der Entwicklungsumgebung beschrieben.

Der Endbenutzer hat die Möglichkeit, die Web-Anwendung mittels eines Browsers aufzurufen. Für die Nutzung der Anwendung werden die folgenden Browser in den angezeigten Versionen vorausgesetzt:

- Chrome v 96
- Firefox v 99
- Safari v 15.6
- Microsoft Edge v 96

Die Entwicklungsumgebung wird auf dem Betriebssystem Windows 10 installiert. In der folgenden Tabelle werden die für die Entwicklung verwendeten Technologien dargestellt:

Komponente	Technologie	Entwicklungssprache
Frontend	Flutter Web	Dart
Backend und API	Express.js, Node.js	JavaScript
Datenbank	MongoDB	-
Visionskontrolle	GitHub	-
Hosting	Localhost	-

Tab. 2: Implementierte Technologien

Als Entwicklungsumgebung wird Visual Studio Code (VSCode) verwendet. Für das Testen der API und die Überprüfung der Verbindung zur API wird die Software Postman eingesetzt. Zur Visualisierung der Datenbank wird die Software MongoDB Compass genutzt.

Im Folgenden wird die Einrichtung der Entwicklungsumgebung beschrieben und in die folgenden Schritte unterteilt:

Nr.	Schritt	Technologie
1	Installation der Entwicklungsumgebung	Visual Studio Code v.1.77 [20]
2	Konfiguration der Entwicklungsumgebung	Extension Dart v3.92.0, Extension Flutter v3.92.0, Extension MongoDB for VS Code v1.6.0.
3	Einrichtung Flutter	Flutter SDK gemäß der Flutter-Dokumentation [21]

4	Einrichtung des Node.js Servers	Node.js Servers gemäß der Server-Dokumentation [22]
5	Einrichtung MongoDB	MongoDB-Datenbank gemäß der Datenbank-Dokumentation [23]

Tab. 3. Einrichtung der Entwicklungsumgebung

Nach Installation der Tools und Frameworks für die Entwicklung folgt in VSCode die Installation der Abhängigkeiten. Dies umfasst den Start des Servers und der Anwendung, welche über das integrierte Terminal in VSCode ausgeführt werden [24]. Eine Übersicht der erforderlichen Schritte, die im Terminal auszuführen sind, wird in der nachfolgenden Tabelle zusammengefasst:

Nr.	Schritt	Beschreibung
1	Installation Abhängigkeiten vom Server	Im Verzeichnis <code>./da-vinci-nodejs-server</code> wird Befehl <code>npm install</code> ausgeführt
2	Installation Abhängigkeiten von Flutter Web	Im Verzeichnis <code>./da_vinci</code> wird Befehl <code>flutter pub get</code> ausgeführt
3	Start des Servers	Im Verzeichnis <code>./da-vinci-nodejs-server</code> wird Befehl <code>npm start</code> ausgeführt
4	Start von Flutter Web	Im Verzeichnis <code>./da_vinci</code> wird Befehl <code>flutter run</code> ausgeführt

Tab. 4: Installation von Paketabhängigkeiten der Entwicklungsumgebung

Die beschriebenen Technologien und Werkzeuge bilden die Basis für die Entwicklung, das Testen und die Ausführung der Anwendung.

## 6.2 Flutter

Im folgenden Kapitel wird das Frontend der Anwendung beschrieben. Zuerst wird eine Bewertung von Cross-Plattform Frameworks zur Auswahl des Frameworks für das Frontend gemäß den Anforderungen des Pflichtenheftes durchgeführt. Anschließend wird die Projektstruktur des Frontends erläutert. Dies umfasst das Design der Benutzeroberfläche (UI), den Zugriff auf die API, die Fehlerbehandlung bei API-Anfragen bzw. -Antworten und das Routing für die Benutzeroberfläche.

### 6.2.1 Vergleich und Begründung

In diesem Kapitel wird die Auswahl eines geeigneten Cross-Platform-Frameworks für die Entwicklung der Anwendung begründet. In Anhang A findet sich eine Statistik von statista.com, welche die am häufigsten verwendeten Cross-Platform-Frameworks für die Jahre 2019 bis 2023 präsentiert. Die Grafik zeigt, dass Flutter und React Native die beiden beliebtesten Frameworks sind. Darüber hinaus ermöglichen beide Frameworks die Umsetzung der im Pflichtenheft aufgeführten Anforderungen. Die Auswahl des Frameworks erfolgt daher durch einen Vergleich dieser beiden Optionen. Der Vergleich basiert auf fünf Kriterien: *Erlernbarkeit, Entwicklungszeit, Design, Performance* und *Routing*. Jedes Kriterium wird auf einer Skala von 1 bis 5 bewertet, wobei 5 die beste Bewertung darstellt. Die Zusammenfassung der Bewertung in der Tabelle wird im Folgenden dargestellt und diskutiert:

Kriterium	Cross-Platform Framework	
	Flutter	React Native
Erlernbarkeit	3	2
Entwicklungszeit	4	3
Design	4	3
Performance	5	3
Routing	5	5
Summe	21/25	16/25

Tab. 5: Vergleich von Flutter zu React Native

Nach einer Analyse der Dokumentation beider Frameworks wurde festgestellt, dass diese gut strukturiert und verständlich sind [21, 25]. Die Syntax von Flutter ist jedoch einfacher zu verstehen und die Integration von Bibliotheken ist zugänglicher, daher die Bewertung von 3 zu 2. Flutter erhält eine bessere Bewertung bei der Entwicklungszeit, da es durch stabile Aktualisierungen des Frontends während der Entwicklung und die Einsetzung von vorgefertigten Widgets eine schnellere Umsetzung ermöglicht, somit die Bewertung von 4 zu 3. Flutter erhält die höchste Punktzahl im Bereich Design, da es eine große Auswahl an gestalteten Widgets bietet, die flexibel und anpassbar sind. React Native bietet ebenfalls gute Designmöglichkeiten, verwendet aber hauptsächlich native UI-Komponenten, was die Anpassungsmöglichkeiten im Vergleich zu Flutter

etwas einschränkt. Flutter bietet eine hervorragende Performance, da der Code für Flutter Web-Anwendungen mittels WebAssembly kompiliert wird. Dies ermöglicht eine nahezu native Ausführungsgeschwindigkeit. Die Performance von React Native ist gut, aber zeichnet sich in einigen Experimenten langsamer gegenüber Flutter aus [16, 17]. Aus diesem Grund ergibt sich eine Bewertung von 5 zu 3. Beide Frameworks bieten leistungsfähige und flexible Lösungen für das Routing, was zu der gleich hohen Bewertung von 5 Punkten führt.

Die Summe der Bewertungen zeigt, dass Flutter mit 21 von 25 Punkten für dieses Projekt besser geeignet ist als React Native mit 16 von 25 Punkten. Insbesondere die hervorragende Performance und die vorgefertigten flexiblen Designmöglichkeiten sprechen für den Einsatz von Flutter.

## 6.2.2 Einführung

Flutter ist ein spezielles Dart-Paket, genannt Plugin-Paket (oder Plugin), das plattform-spezifischen Code enthält. Das Framework bietet die Möglichkeit mit einer Code-Basis und mit überschaubaren Anpassungen eine Webanwendung, eine Android App und eine iOS-App zu kompilieren. Aufgrund der Anforderungen ist geplant in Zukunft eine App für die Logistik zu erstellen und die Funktionalität durch eine Warenverfolgung zu erweitern.

Für die Darstellung des Widgets verwendet Flutter die Rendering Engine Skia. Skia ist eine leistungsfähige Open-Source-Grafikbibliothek, die für die Erstellung von 2D-Grafiken zuständig ist. Die Verwendung von Skia hat einen großen Einfluss auf die Gesamtperformance der Anwendung. Skia arbeitet plattformübergreifend und verbindet sich direkt mit der Schnittstelle des Betriebssystems des Geräts [26].

Die Verwendung der Programmiersprache *Dart* ermöglicht es Flutter, den Quellcode frühzeitig in nativen Code zu kompilieren und erlaubt ein schnelles Starten und Ausführen der App während der Lauf- und Entwicklungszeit.

Das Framework ist mit Material Design und Cupertino Libraries ausgestattet. Material Design ist eine von Google entwickelte Open Source Bibliothek, die das Styling der Komponenten und das Aussehen der Webanwendung gestaltet. Sie bietet zahlreiche vorgefertigte Designlösungen.

Des Weiteren stellt Flutter eine PUB-Paketmanager-Funktion zur Verfügung, über die externe Module oder Pakete für bestimmte Programmierlösungen installiert werden

können. Der Einsatz externer Module erlaubt eine Beschleunigung des Entwicklungsprozesses [14].

### 6.2.3 Struktur

Die Struktur des Flutters-Projekts ist in drei Schichten unterteilt:

- Benutzeroberfläche und Anwendungslogik
- Datenzugriffsschicht
- Datenmodellschicht

Die implementierte Benutzeroberfläche besteht aus einer Vielzahl von Widgets, deren Standarddesign an die spezifischen Anforderungen der Anwendung angepasst wird. Widgets sind unveränderliche Objekte, die die Struktur, das Layout und das Verhalten von UI-Komponenten definieren. Flutter ermöglicht es, komplexe und ansprechende Benutzeroberflächen durch die Kombination von Widgets zu erstellen. Die Anwendungslogik ist eng mit der Benutzeroberfläche verbunden, sodass Benutzerinteraktionen direkt in die Logik der App integriert werden können. Dies umfasst die Behandlung von Benutzerereignissen, die Navigation innerhalb der Anwendung und die dynamische Anpassung der Benutzeroberfläche.

Die Datenzugriffsschicht ist für die Kommunikation mit dem Server zuständig. Sie verwaltet HTTP-Anfragen, kümmert sich um die Authentifizierung der Anfragen und verarbeitet die Antworten des Servers. Diese Schicht ist von der restlichen Anwendungslogik entkoppelt, um die Testbarkeit und Wartbarkeit der App zu verbessern. Durch die klare Trennung der Datenzugriffsschicht kann der Zugriff auf externe APIs isoliert werden, was es einfacher macht, diese Komponenten zu testen und bei Bedarf auszutauschen. Diese Schicht stellt auch die Methoden bereit, die den API-Zugriff ermöglichen und somit die Grundlage für den Datenaustausch mit dem Server bilden.

In der Datenmodellschicht werden Klassen definiert, die vom Server empfangene Daten repräsentieren. Die Datenmodellschicht ermöglicht es, die empfangenen Daten in geeigneter Weise zu verarbeiten und für die Anwendung nutzbar zu machen, beispielsweise durch die Konvertierung von JSON-Antworten in Dart-Objekte. Durch die Strukturierung der Daten in dieser Schicht wird eine klare und konsistente Handhabung der Daten im gesamten Frontend sichergestellt.

#### 6.2.4 Design der Benutzeroberfläche der Anwendung.

Die Gestaltung der Benutzeroberfläche erfolgt unter Verwendung des Standardthemas *Material 3* [14]. Hierbei handelt es sich um ein Open-Source-Designsystem von Google. Das Designsystem definiert das vollständige Aussehen der Anwendung. Im Rahmen der Umsetzung von nichtspezifischen Anforderungen des Pflichtenhefts bezüglich der Benutzeroberfläche erfolgt eine Anpassung des Themes aus *Material 3*. Dazu wird eine Klasse mit statischen Farbeigenschaften in der gesamten Anwendung implementiert. Die genannten Farben definieren die Schaltflächenfarben, Hintergrundfarben und Textfarben. Im Rahmen der Anpassung des Themes wird eine Instanz des *ThemeData*-Objekts erzeugt. Die Klasse *ThemeData* beinhaltet die Farb- und Typografiewerte für ein Material-Design-Theme. Die vordefinierten Farben werden sowohl als globale Einstellungen als auch in den einzelnen Widgets verwendet.

Die Anwendung weist ein responsives Design auf, welches eine Adaption an die jeweilige Bildschirmgröße ermöglicht. Zu diesem Zweck wird ein Breakpoint bei einer Breite von 768 ppi implementiert. Zur Ermittlung der Bildschirmgröße wird auf die von Flutter bereitgestellte Klasse *MediaQuery* zurückgegriffen. Die Abb. 4, 5, 6 und 7 präsentieren das Design der Anwendung.

Somit werden die nichtspezifischen Anforderungen, bezüglich des Designs, an das System erfüllt. Die in den Abbildungen 4-7 dargestellten Menüs und Layouts verdeutlichen die Umsetzung dieser Designprinzipien.

Der Benutzer initiiert den Anmeldevorgang über das Frontend, das eine POST-Anfrage an die Route */api/signin* der API sendet. Die API empfängt die Anfrage und versucht, den Benutzer im Backend zu finden. Wenn der Benutzer gefunden wird, überprüft die API das Passwort. Bei erfolgreicher Passwortüberprüfung wird ein Authentifizierungs-Token generiert. Das System aktualisiert den Benutzerdatensatz im Backend (z.B. letzten Anmeldezeitpunkt). Das Backend führt die Aktualisierung durch und gibt die Ergebnisse an die API zurück. Schließlich erhält das Frontend den generierten Token, die E-Mail und den Benutzertyp.

Der Token wird im lokalen Speicher des Browsers gespeichert, um für zukünftige Anfragen verwendet zu werden. Der Authentifizierungs-Token wird im lokalen Speicher im Browser des Nutzers angelegt. Mit Hilfe von Pub-Paket *shared\_preferences* wird auf den lokalen Speicher der Browser zugegriffen. Der Token wird unter dem eindeutigen Schlüssel *token* im Browser angelegt. Der Authentifizierungs-Token wird bei jeder

HTTP-Anfrage an den Server aus dem lokalen Speicher gelesen und in dem Header der Anfrage übergeben. Bei der Ausloggen der Benutzer aus dem System wird der Token aus dem Speicher gelöscht. Somit wird das Überlaufen der lokalen Speicher sichergestellt.

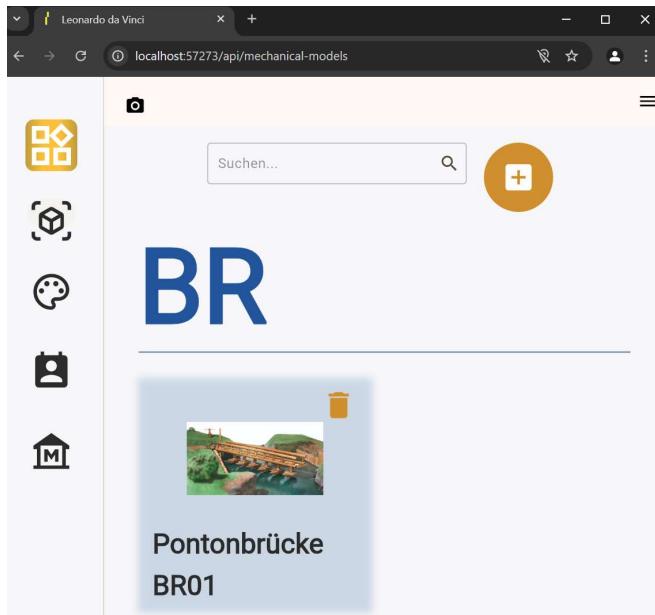


Abb. 4: Hauptfenster nach der Anmeldung im System. Ansicht auf dem Desktop-PC.

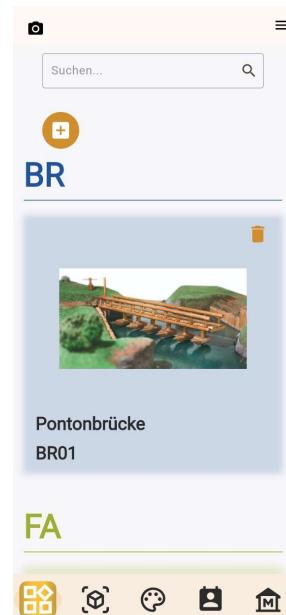


Abb. 5: Hauptfenster nach der Anmeldung im System. Ansicht auf dem Handy.

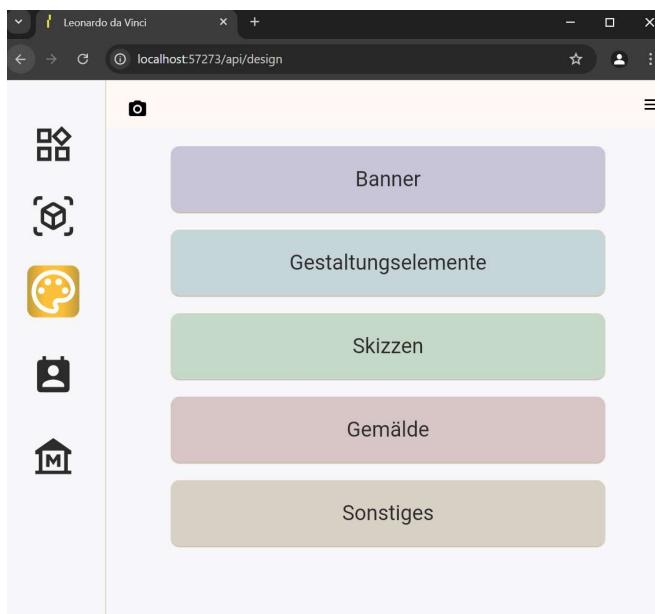


Abb. 6: Hauptansicht für Gestaltungselemente. Ansicht auf dem Desktop-PC.



Abb. 7: Hauptansicht für Gestaltungselemente. Ansicht auf dem Handy.

## 6.2.5 HTTP-Anfragen an die API

In diesem Abschnitt wird beschrieben wie die HTTP-Anfragen an die API zu senden sind und wie der Inhalt von HTTP-Anfragen korrekt formatiert wird. Um API-Aufrufe zu

ermöglichen, wird das *http*-Paket aus dem Pub-Management-System installiert. Das Paket unterstützt verschiedene HTTP-Operationen wie GET, POST, PUT, DELETE usw. und kann sowohl synchrone als auch asynchrone Anfragen verarbeiten.

Damit der Server eine Anfrage vom Client akzeptiert, muss die Anfrage in einem von dem Server bestimmten Format gesendet werden, die im Kapitel 6.4.3 beschrieben sind. Somit muss der HTTP-Header folgende Informationen beinhalten:

- Autorisierungs-Header im Format: *Authorization: Bearer token*
- Repräsentations-Header *Content-Type*:

Der Autorisierungs-Header wird verwendet, um den Zugriff auf den Server zu ermöglichen. Der Repräsentations-Header wird verwendet, um den ursprünglichen Medientyp der Ressource anzugeben, bevor eine Inhaltskodierung vor der Übertragung vorgenommen wird. In der Anwendung werden zwei Medientypen verwendet, die das Format des Inhaltes beschreiben [27]:

- *Content-Type: application/json; charset=UTF-8*
- *Content-Type: multipart/form-data*

Die *http*-Bibliothek stellt die Klasse *Client* zur Verfügung, um eine dauerhafte Verbindung zum Server zu ermöglichen. Für http-Anfragen wird ein benutzerdefinierter HTTP-Client implementiert, der so konzipiert ist, dass er automatisch Authentifizierungs-Token verarbeitet und geeignete Content-Type-Header für ausgehende HTTP-Anfragen setzt.

Mit Hilfe des Pakets *fpdart* wird die Fehlerbehandlung während der http-Aufrufe durchgeführt. Es bietet spezielle Typen wie *Either*, die verwendet werden, um zwischen Erfolgs- und Fehlerzuständen zu unterscheiden. Im Fall des Erfolgs wird eine *Right* Instance zurückgegeben, ansonsten eine *Left* Instance, die eine Klasse für die Fehlerbehandlung *Failure* bekommt. Die Klasse *Failure* kapselt einen Fehlerzustand und enthält eine Nachricht zur Beschreibung des Fehlers, die als Antwort von der API gemeldet werden sowie einen optionalen StackTrace zur Bereitstellung von Debugging-Informationen. Der Fehler wird dann als Nachricht in einer Statusleiste im Browser dargestellt. Somit wird die Anforderung /B01/ erfüllt.

## 6.2.6 Navigation in der Anwendung

Das vorliegende Kapitel dient der näheren Erläuterung des in der Anwendung integrierten Navigationssystems. Die Implementierung des Navigationssystems verfolgt das Ziel, eine nutzerfreundliche Gestaltung und eine intuitive Nutzung der Anwendung zu ermöglichen. Die Navigation basiert auf dem Paket `go_router`, welches eine verschachtelte Navigation ermöglicht. Das Navigationssystem der Anwendung wird in Abb. 7 grafisch dargestellt.

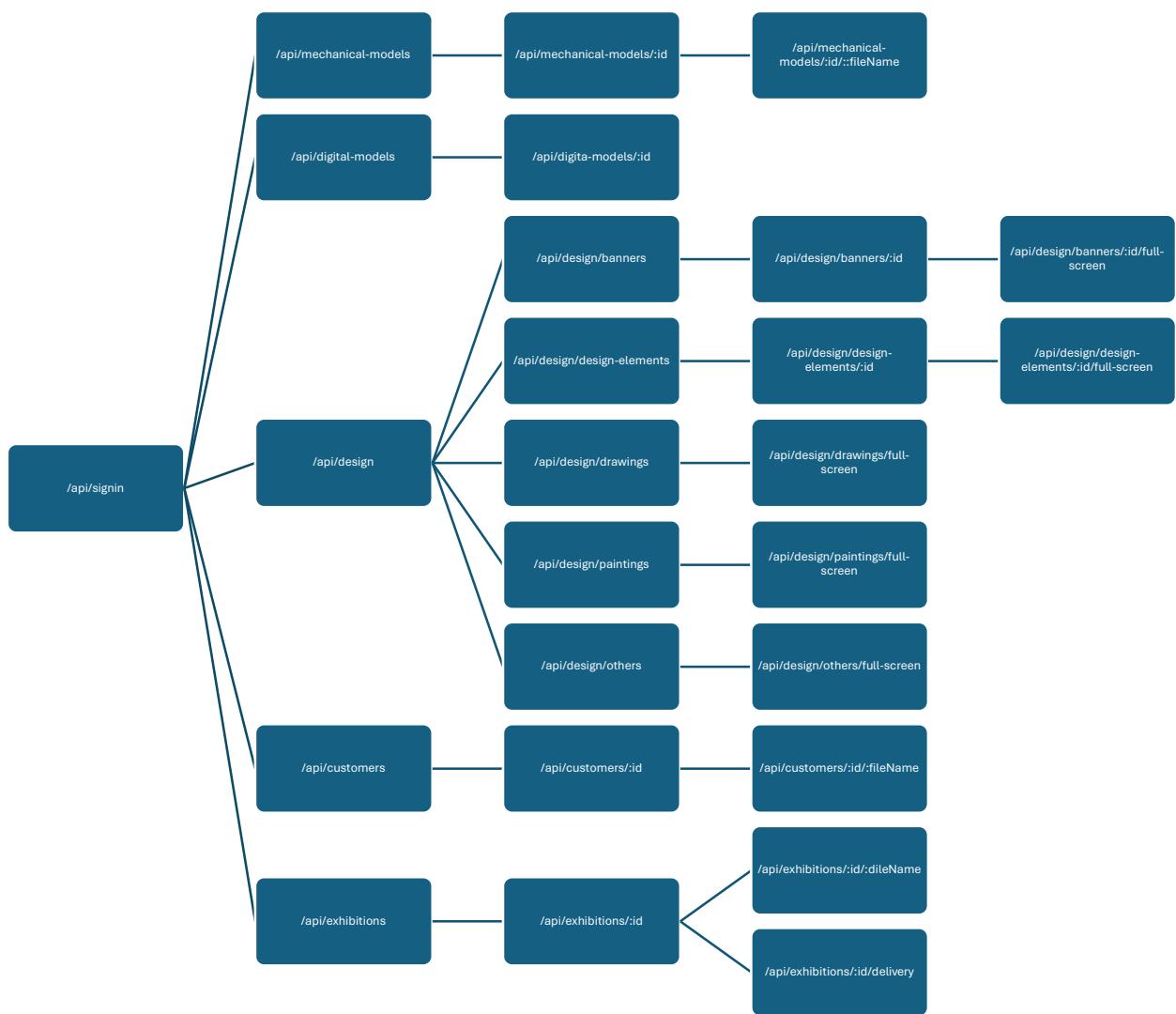


Abb. 8: Navigationssystem der Anwendung im Frontend.

Das System setzt sich aus unabhängigen Navigationszweigen und dem Anmeldepfad zusammen. Die Navigation startet bei der Anmelderoute in der ersten Säule und fünf Hauptrouten, die in der zweiten Säule abgebildet sind. Weitere Säulen veranschaulichen die verschachtelten Routen innerhalb der zuvor erwähnten Hauptrouten. Die Navigation ist so konzipiert, dass zunächst eine Liste mit Elementen einer Gruppe angezeigt wird. Bei Auswahl eines Elements werden dessen Details auf einem weiteren Bildschirm dargestellt. In den Endrouten erfolgt die Anzeige von Dokumenten oder Bildern im Vollbildmodus. Bei einer nicht vorhandenen Pfadangabe wird der Benutzer zunächst zu einer Fehlerseite weitergeleitet und kann anschließend wieder zum Hauptmenü zurückkehren. Im Rahmen der Gestaltung der Benutzeroberfläche werden zwei unterschiedliche Varianten für die Darstellung der Menüleiste entwickelt.

In den Abb. 4 und 5 sind die Hauptansichten der Anwendung dargestellt. Diese umfassen ein Menü, eine Kopfzeile sowie eine Fläche für das Content-Management. Für Geräte mit einem kleinen Bildschirm bzw. einer Auflösung bis zu 768 ppi. wird ein horizontales Menü am unteren Rand des Bildschirms erstellt. Bei einer Auflösung von 768 ppi oder mehr erscheint ein vertikales Menü am linken Rand. Die Schaltflächen im Menü repräsentieren jeweils eine Hauptroute und bieten Zugang zu mech. Modellen, virtuellen Modellen, Gestaltungselementen sowie Kunden und Ausstellungen.

In der Web-Applikation wurde ein komplexes Navigationssystem mit mehreren unabhängigen Navigationsbäumen implementiert, welches sich durch eine intuitive Benutzerfahrung auszeichnet. Das System ist skalierbar und ermöglicht eine flexible Organisation von Bildschirmen sowie eine strukturierte Navigationslogik innerhalb der Anwendung.

### 6.2.7 Produktfunktionen

Die im Pflichtenheft beschriebenen Produktfunktionen des AMS wurden in der Implementierung detailliert und systematisch umgesetzt. Dieses Kapitel beschreibt, wie die einzelnen Funktionen realisiert wurden, um den Anforderungen des Systems gerecht zu werden. Dabei werden, aufgrund des Umfangs, nur einige Beispiele angeführt.

## /PF20/ Benutzer anlegen

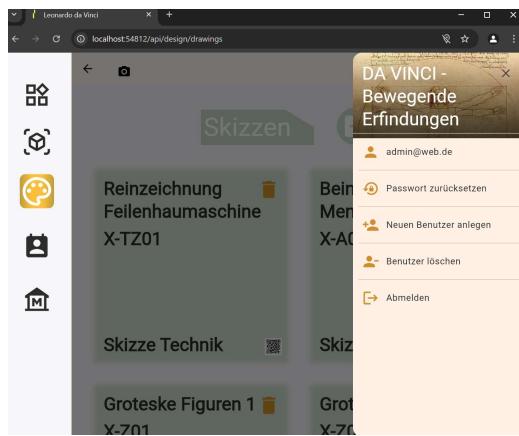


Abb. 9: Benutzerverwaltung Admin

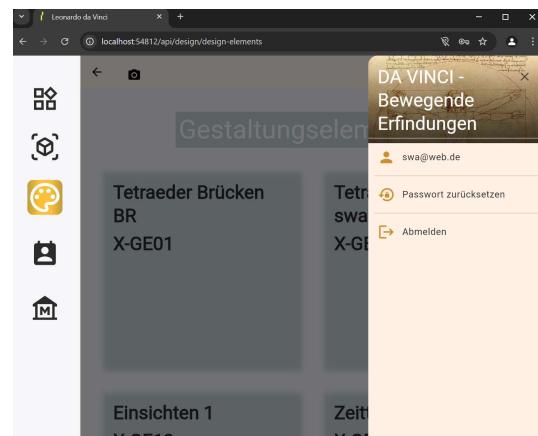


Abb. 10: Benutzerverwaltung. SWA und WMA

This screenshot shows a dialog box titled 'Neues mechanisches Modell' (New mechanical model). The form contains the following fields:

- Bezeichnung (Description)
- Kürzel (Abbreviation)
- Familie (Family) dropdown menu
- Länge in cm (Length in cm)
- Breite in cm (Width in cm)
- Höhe in cm (Height in cm)
- Gewicht in kg (Weight in kg)
- Anzahl (Quantity)
- Versicherung in € (Insurance in €)
- Checkboxes for categories: Strom (Strom), Wasser (Wasser), Bedienpersonal (Bedienpersonal), and Aufsteller Schild (Aufsteller Schild).
- Betätigung (Operation) dropdown menu
- Modellelement (Model element) dropdown menu
- Modellkarte (Model map) checkbox

Abb. 11: Dialogfenster zur Erstellung von einem neuen Vermögenswert.

This screenshot shows a detailed view of a mechanical model entry. The main card displays the following information:

- Sockel** (+ icon)
- Zuberhör** (+ icon)
- Verpackung** (Packaging icon)
- Bezeichnung: Verpackung Fallschirm
- Verpackungsart: Sonstiges
- Länge: 110cm
- Breite: 110cm
- Höhe: 200cm
- Volumen: 2.42 m³
- Stapelbar: Nein
- QR code
- Palette: Europalette

Below this card is another blue button labeled 'Dokumente' with a plus sign icon.

Abb. 12: Ausschnitt der Detailansicht eines mechanischen Modells.

## /PF50/ Vermögenswert erstellen

Die Erstellung von Vermögenswerten ist eine zentrale Funktion des mobilen AMS. Benutzer mit entsprechenden Rechten können neue Vermögenswerte anlegen, indem sie relevante Informationen wie Bezeichnung, Kategorie, und andere Attribute eingeben. Nach der Eingabe generiert das System einen eindeutigen QR-Code für den Vermögenswert, der in der Datenbank gespeichert wird. Diese Funktionalität wurde durch die Implementierung eines flexiblen Formularsystems und einer QR-Code-Generierungs-API realisiert. Die Produktfunktionen /PF80/, /PF90/, /PF100/, /PF110/, /PF120/ erweitern /PF50/ und werden mit dem gleichen Design umgesetzt.

## /PF170/ Angebot erstellen

Benutzer können Angebote für Kunden erstellen, indem sie entsprechende Modelle und Gestaltungselemente auswählen und den Angebotszeitraum festlegen. Das Aktivitätsdiagramm stellt grafisch die umgesetzte Produktfunktion dar.

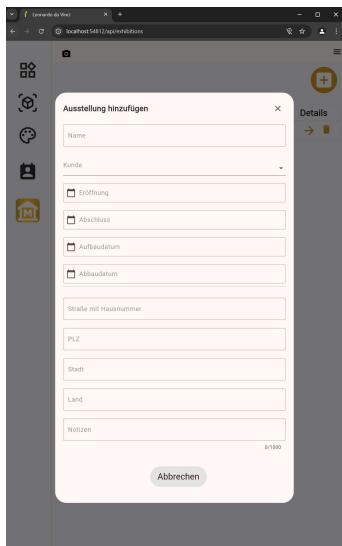


Abb. 13: Das Dialogfenster, um eine neue Ausstellung zu erstellen.



Abb. 14: Dialogfenster, um eine neue Ausstellung zu erstellen, nachdem der Kunde ausgewählt wurde.

Abb. 15: Benutzeroberfläche für die Erstellung eines Angebots der Kunde

Abb. 16: Erweiterung des Angebots um mechanische Modelle

Abb. 17: Angebot des Kunden, nachdem die Vermögenswerte zur Angebotsliste hinzugefügt wurden.  
Das System ist bereit die Angebotsliste zu speichern und wartet auf die Aktion des Benutzers.

Abb. 18: Das Angebot des Kunden ist erstellt. Eine neue Ausstellung kann angelegt werden.

## /PF220/ Ausstellung erstellen

Benutzer können neue Ausstellungen erstellen, indem sie relevante Daten eingeben und diese einem Kunden zuordnen. Das Aktivitätsdiagramm beschreibt den Prozess.

## 6.3 Progressive-Web-App

Ein weiterer Schwerpunkt liegt auf der Realisierung der PWA, bei der Funktionen wie QR-Code-Scanning, Kameranutzung und Geolokalisierung integriert werden. Im Folgenden wird detailliert beschrieben, wie die QR-Code- und Kameraverwendung in der PWA realisiert wurde.

### 6.3.1 QR-Code und Kameraverwendung

In der App ist die Erstellung des QR-Codes integriert. Der QR-Code ist ein zweidimensionaler Code. Die Erstellung des QR-Codes kann sowohl im Frontend als auch im Backend stattfinden. In der Anwendung wird der QR-Code im Backend generiert. Im Frontend kann der Benutzer den QR-Code mit Hilfe der Kamera auslesen oder runterladen. Mit einem Klick auf den QR-Code wird dieser in einem weiteren Browser-Fenster geöffnet. Dafür wird das Paket *url\_launcher* benötigt. Das Paket ermöglicht das Öffnen des Standardbrowsers auf der mobilen Plattform, um eine bestimmte URL anzuzeigen. Mit einem Rechtsklick auf den QR-Code und der Auswahl des Elements *Bild speichern unter...* wird der QR-Code in einem benutzerdefinierten Pfad auf dem System des Benutzers gespeichert.

Für das Scannen der QR-Codes wird das Paket *mobile\_scanner* verwendet. Es integriert den Zugriff auf die Kamera und das Scannen des QR-Codes. Die Abb. 21 und 22 bilden die Funktionalität der Kamera ab.

Das Scannen des QR-Codes wird durch das Betätigen der Schaltfläche mit dem Kamera-Symbol. Der QR-Code wird für die folgende Elemente generiert und kann unter folgenden Pfaden ausgelesen werden:

- Mechanische Modelle, Sockel, Zubehör, Verpackung: */api/mechanical-model*
- Virtuelle Modelle: */api/digital-models*
- Banner: */api/design-elements/banners*
- Skizzen: */api/design-elements/design-objects*
- Gemälde: */api/design-elements/drawings*
- Gestaltungselemente: */api/design-elements/paintings*
- Gestaltungobjekte: */api/design-elements/others*

Die Wunschanforderung, dass jedem physischen Asset ein eindeutiger QR-Code zugewiesen wird, ist erfüllt. Die Wunschanforderung /PFW250/ nach dem Scannen des

QR-Codes wird nicht erfüllt. Die vollständige Implementierung dieser Funktionalität konnte nicht realisiert werden. Die Funktionalität des Scannens von QR-Codes führte in einigen Fällen zu einem Stillstand des Systems, sodass eine weitere Überarbeitung erforderlich ist, um die Effektivität und Zuverlässigkeit zu gewährleisten.

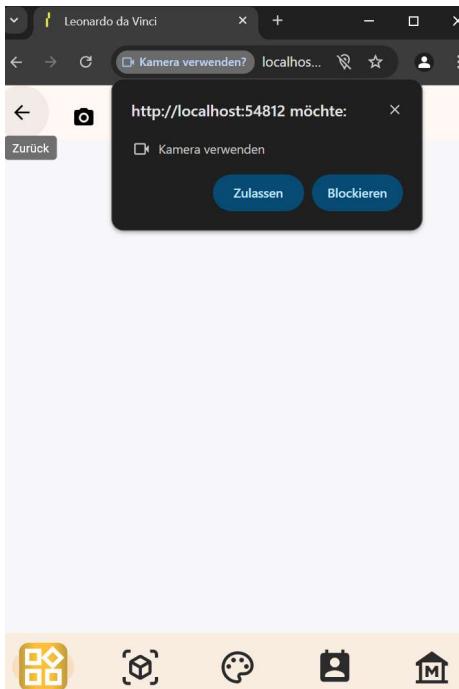


Abb. 19: Zugriff auf die Kamera.

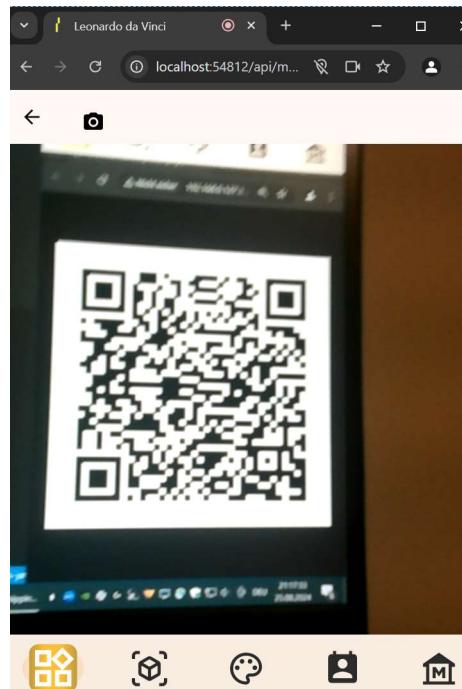


Abb. 20: Kamera in aktivem Zustand und ist für die Scannung der QR-Codes bereit.

### 6.3.2 Geolokalisierung

Analog zu den grundlegenden Eigenschaften progressiver Web-Anwendungen kann die vorliegende Anwendung potenziell Zugriff auf GPS-Daten des Benutzers erhalten. Im Folgenden erfolgt eine Erläuterung des Geolokalisierungsprozesses. Der Begriff **Geolokalisierung** bezeichnet den Prozess der Bestimmung des tatsächlichen Standortes eines Geräts oder Nutzers. Die Lokalisierung kann dabei auf Basis verschiedener technologischer Parameter erfolgen, darunter GPS, WLAN-Signale, IP-Adressen oder Mobilfunknetzdaten. Das Global Positioning System (GPS) stellt ein satellitengestütztes Navigationssystem dar, welches GPS-Empfängern an jedem Ort der Erde oder in deren Nähe genaue Standort- und Zeitinformationen bereitstellt. Die Berechnung der geografischen Koordinaten (Breiten- und Längengrad) erfolgt durch Triangulation der Signale von mehreren Satelliten. In Bezug auf die Auslesung der GPS-Daten der Benutzer besteht grundsätzlich die Möglichkeit, diese sowohl im Frontend als auch im

Backend vorzunehmen. Das Flutter Framework und Node.js bieten eine Vielzahl von Möglichkeiten, auf die GPS-Daten der Benutzer zuzugreifen.

Die Steuerung der Anwendung erfolgt durch den Browser, wodurch diesem die volle Kontrolle über die freigegebenen Daten sowie die blockierten Daten obliegt. Die Freigabe der GPS-Daten erfolgt erst nach einer entsprechenden Benutzeranfrage. Die Auslösung erfolgt mittels unterschiedlicher Pakete in Flutter. Für die Extraktion von GPS-Daten wird das Paket *location* in der Version 6.0.2 installiert.

Bei Aufruf der Funktionen zur Standortbestimmung wird im Browser ein Pop-up-Fenster angezeigt, in dem nach Freigabe der Standortdaten gefragt wird. Im Anschluss an die Auslesung beider Werte für den Standort erfolgt eine Konvertierung zu einer Adresse. Für die Konvertierung der GPS-Daten wäre ein weiteres Paket von Flutter oder die Verwendung einer fremden API erforderlich. Die Konvertierung der GPS-Daten zu einer Adresse wurde nicht realisiert. Somit ist die Wunschanforderung an die Standorterfassung der Assets nicht umgesetzt, da das System nur die GPS-Daten abrufen und speichern kann, diese aber abschließend nicht zu einer Adresse konvertiert werden. Die Abb. 23 zeigt die Umsetzung der Berechtigungsanfrage für den Standort in der Anwendung.

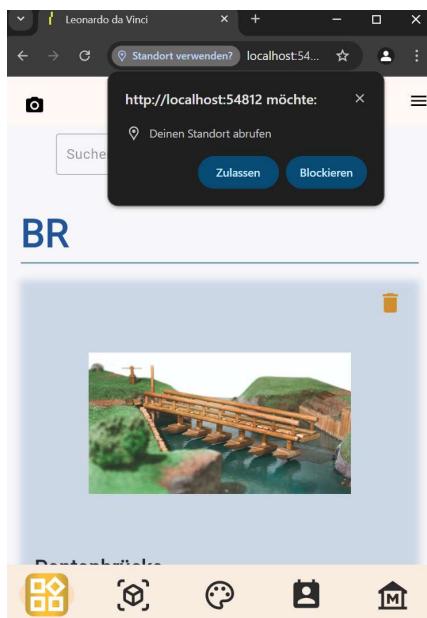


Abb. 21: Standortüberwachung des Benutzers.

Im Rahmen der Erstellung oder dem Editieren der Daten für mechanische Modelle und Gestaltungselemente besteht für den Benutzer die Möglichkeit, die Breiten- und Längengrade sowie den Standort zu hinterlegen. Die Voraussetzung hierfür ist die Freigabe der GPS-Daten im Browser.

## 6.4 MongoDB

Um die Anforderungen der Anwendung zu erfüllen, muss die Datenbank eine hohe Flexibilität und eine durchschnittliche Datenkonsistenz aufweisen. Die hohe Flexibilität ist notwendig, da sich die Anforderungen an das Projekt über die Projektdauer schnell ändern können. Eine durchschnittliche Datenkonsistenz ist ausreichend, da es sich bei der Anwendung, um ein Asset-Management-System der Kunstbranche handelt. Fehler in den Daten führen hier nicht zu schwerwiegenden Personenschäden führen, wie es bspw. In der Automobil- oder Medizinbranche der Fall sein könnte. Außerdem erfordert das Projekt eine sehr schnelle Entwicklungszeit, da während des Projektes insgesamt nur einige Monate für die Entwicklung zur Verfügung stehen.

### 6.4.1 Vergleich von Datenbanktechnologien

Für die oben genannten Kriterien kommen insbesondere zwei Technologien in Frage. Beide Technologien haben spezifische Stärken und Schwächen, die sich in verschiedenen Anwendungsfällen unterschiedlich auswirken. In diesem Kapitel werden beide Datenbanktypen verglichen und es werden Gründe für die Wahl für eine von beiden Technologien erläutert.

Der Vergleich basiert auf drei Kriterien: *Flexibilität*, *Konsistenz* und *Entwicklungszeit*. Jedes Kriterium wird auf einer Skala von 1 bis 5 bewertet, wobei 5 die beste Bewertung darstellt. Die Zusammenfassung der Bewertung wird in der folgenden Tabelle dargestellt und im Anschluss diskutiert:

Kriterium	Datenbanktyp	
	SQL	NoSQL
Flexibilität	2	5
Konsistenz	5	3
Entwicklungszeit	3	5
Summe	10/15	13/15

Tab. 6: Vergleich der Datenbanktechnologien

SQL-Datenbanken setzen eine statische Struktur voraus, sodass Änderungen mit viel Aufwand verbunden sind. NoSQL-Datenbanken hingegen sind schemilos und dadurch wesentlich flexibler. Daher wurde hier eine Bewertung von 2 zu 5 vergeben.

SQL-Datenbanken folgen dem ACID-Prinzip (Atomicity, Consistency, Isolation, Durability), um eine hohe Konsistenz der Daten gewährleisten zu können. Jede Transaktion führt das System von einem konsistenten Zustand in einen anderen, was bedeutet, dass die Integrität der Daten jederzeit gewährleistet ist. NoSQL-Datenbanken setzen oft auf das BASE-Prinzip (Basically Available, Soft State, Eventual Consistency), bei dem die strikte Einhaltung eines festen Konsistenzmodells entfällt. Die Daten sind nicht sofort konsistent über alle Knoten hinweg, sondern erreichen schließlich einen konsistenten Zustand. Somit die Bewertung 5 zu 3. SQL-Datenbanken benötigen gründliche Planung, was zu einer längeren Entwicklungszeit führen kann. Hingegen bieten NoSQL-Datenbanken eine schnellere Entwicklungszeit, insbesondere in der frühen Phase des Projekts, dank ihrer Struktur, die schnelle Änderungen und Anpassungen erlaubt. Dies rechtfertigt eine Bewertung von 3 zu 5 [28].

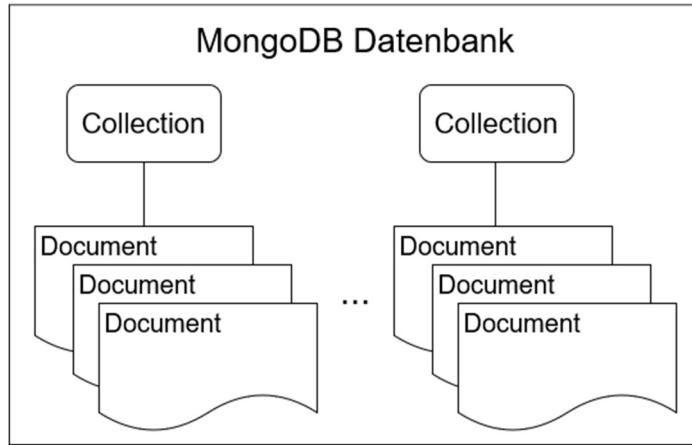
Die Summe der Bewertungen zeigt, dass NoSQL mit 13 von 15 Punkten für dieses Projekt besser geeignet ist als SQL mit 10 von 15 Punkten. Insbesondere die Flexibilität und schnelle Entwicklungszeit der Datenbank sprechen für den Einsatz von NoSQL.

Im nächsten Schritt erfolgt die Auswahl eines geeigneten Datenbankmanagementsystems (DBMS). DBMS ist eine Software zur Steuerung, Verwaltung und Kontrolle der in der Datenbank gespeicherten Daten. Das DBMS führt die Zugriffe des Anwenders oder der Anwendungssoftware (z.B. Lesen, Anlegen, Ändern oder Löschen von Datensätzen) über eine Kommunikationsschnittstelle aus. Im Anhang B findet sich eine Statistik von statista.com, welche die am häufigsten verwendeten DBMS weltweit im Jahr 2022 präsentiert. Die Grafik zeigt, dass die ersten vier DBMS SQL-Datenbanken verwalten. Auf dem fünften Platz steht MongoDB, die dem NoSQL-Datenbanktyp zugeordnet wird. Somit kommt MongoDB für die Entwicklung der Datenbank zum Einsatz.

#### 6.4.2 Aufbau

MongoDB ist eine dokumentbasierte NoSQL-Datenbank. Die Datenbank organisiert Daten in flexiblen BSON-Dokumenten, welche wiederum in mehreren Sammlungen innerhalb der Datenbank zusammengefasst werden. BSON (Binary JSON) ist ein binäres Format, das zum Speichern von Dokumenten dient und die JSON-Syntax für die Übertragung von Daten verwendet. JSON (JavaScript Object Notation) ist ein standardisiertes textbasiertes Format zur Darstellung strukturierter Daten auf der Grundlage der JavaScript-Objektsyntax. Das BSON-Format ermöglicht einen schnelleren Lese-

und Schreibzugriff auf Daten, was insbesondere bei großen Datenmengen zu einer Leistungssteigerung führt [28]. Abb.5 präsentiert die Aufbaustruktur der Datenbank, welche sich aus Collections und Documents zusammensetzt. Die Collection repräsentiert eine Sammlung von Documents. Die Datenbank kann eine beliebige Anzahl an Collections aufweisen, während eine Collection eine beliebige Anzahl an Documents beinhalten kann.



*Abb. 22: Struktur der implementierten Datenbank.*

Im Folgenden wird die Datenmodellierung in MongoDB näher erläutert. Die Datenmodellierung beschreibt die Organisation und Verknüpfung von Daten innerhalb einer Datenbank. Ein Document besteht aus einem Satz von Attributen, die ein Objekt repräsentieren. Documents innerhalb einer Collection müssen nicht denselben Satz von Attributen enthalten. Auch der Datentyp eines Attributs kann sich zwischen den Documents innerhalb einer Collection unterscheiden. Die Attribute sind als Schlüssel-Wert-Paare organisiert. Documents werden durch ein einzigartiges Attribut `_id` identifiziert, das vom MongoDB-System automatisch generiert wird. Dieser Identifikator dient als Primärschlüssel und stellt sicher, dass jedes Document in der Collection eindeutig ist. Alle Attribute eines Documents werden in obligatorische und optionale Felder unterteilt. Beim Anlegen des Objekts in der Datenbank müssen die obligatorischen Attribute angegeben werden. Im Allgemeinen haben Documents in einer Collection eine ähnliche Struktur. Um die Konsistenz des Datenmodells zu gewährleisten, können Schemavalidierungsregeln erstellt werden.

Bei dem Aufbau von Beziehungen in MongoDB es ist wichtig einzuhalten, dass zusammengehörige Daten auch gemeinsam gespeichert werden. Bei der Erstellung von Documents und Collections ist von entscheidender Bedeutung, welche Anwendung die Daten nutzen wird, nicht jedoch, auf welche Weise sie gespeichert werden.

Im Folgenden werden die verschiedenen Beziehungen zwischen Dokumenten erläutert. Dabei wird zwischen den folgenden Arten unterschieden:

- Eine One-to-One-Beziehung liegt vor, sofern ein Document exakt mit einem anderen Document verknüpft ist.
- Eine One-to-Many-Beziehung impliziert, dass ein Document mit mehreren anderen Documents verknüpft ist.
- Eine Many-to-Many-Beziehung bezeichnet eine Verknüpfung von mehreren Documents mit mehreren anderen Documents.

In MongoDB gibt es zwei Möglichkeiten Beziehungen zwischen Documents abzubilden. Die Methoden *Embedding* und *Referencing* dienen der Einbindung externer Daten in ein MongoDB-Dokument. Bei der *Embedding* werden zusammengehörige Daten in das Dokument integriert. Die *Referenzierung* bezeichnet den Verweis auf eine andere Kollektion innerhalb eines Dokuments. Dabei wird das ID-Feld eines Document in einem anderen als Link zwischen zwei Documenten gespeichert. Die *Referenzierung* wird eingesetzt, um die Datenintegrität zu gewährleisten, insbesondere bei der gemeinsamen Nutzung von Daten durch mehrere Dokumente.

Die *Embedding*-Methode findet Anwendung bei One-to-Many- und Many-to-Many-Beziehungen. Die Speicherung von verwandten Daten in einer einzigen Dokumentstruktur, anstatt ihre Verteilung auf verschiedene Dokumente oder Collections, erfolgt durch die Verwendung der Embedding-Methode. Im Rahmen dieser Methode kann ein Dokument Arrays und Objekte mit Bezugsdaten enthalten. Das Datenmodell ermöglicht es Anwendungen, zusammenhängende Daten in einer einzigen Datenbankoperation abzurufen, wodurch die Notwendigkeit entfällt, auf andere Collections zu verweisen. Dies resultiert in einer gesteigerten Performance bei Leseoperationen.

Die Leseperformance einer Operation wird zudem durch die Größe eines Documents beeinflusst, welches als eine Einheit ausgelesen wird. Bei der Konzeption des Datenbankmodells sollte beachtet werden, dass die maximale Größe des BSON-Dokument 16 MB nicht überschritten wird.

#### 6.4.3 Umsetzung der Datenbank

Im Folgenden wird die Umsetzung der Datenbankmodells für das System erläutert. Bei der Modellierung der Datenbank wurden zuerst die Objekte definiert, die Hauptreger der Daten. Zu diesen Objekten gehören mechanische Modelle und Virtuelle Modelle,

Banner, Skizzen, Gemälde, Gestaltungselemente, Gestaltungobjekte, Sockel, Accessoire, Verpackung für das mechanische Modell, Modelfamilien, Kunden, Ausstellungen, Codices, Seiten und Benutzer. Anschließend wurden Objekte zu Collections gruppiert und die Verbindungen zwischen den Collections festgelegt. Daraus entstand zehn Collection, die in der Abb.21 in dem Entity-Relationship-Modell (ER-Modell) der Datenbank dargestellt sind. Das ER-Modell dient zu Visualisierung und Darstellung der Beziehungen zwischen den Collections. Im Folgenden wird das ER-Modell erläutert.

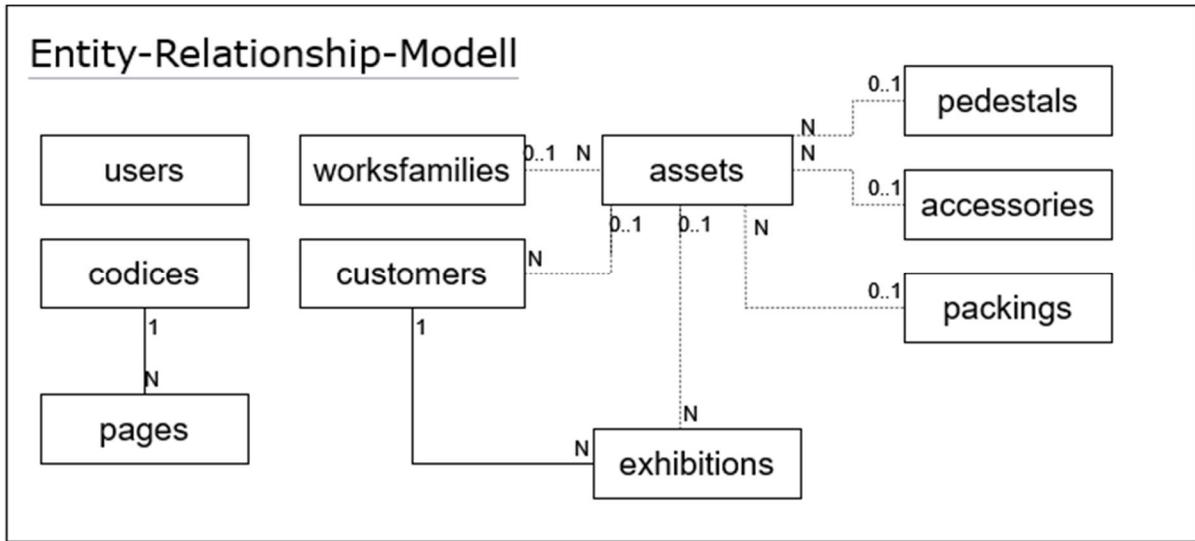


Abb. 23: Entity-Relationship-Modell.

Das ER-Modell besteht aus zehn Collections. Die Verbindungen zwischen den Collections sind mit durchgezogenen oder gepunkteten Linien gekennzeichnet. Die gepunktete Linie stellt eine optionale Beziehung abbildet. Das zentrale Collection **assets** verwaltet Vermögenswerte des Systems. Unter diese Collection befinden sich die Documents für mechanische und virtuelle Modelle, Banner, Skizzen, Gemälde, Gestaltungselemente und Gestaltungobjekte. Die Collection **assets** weist optionale One-to-Many Verbindungen zu den **pedestals** (Sockel), **accessories** (Zubehör) und **packings** (Modellverpackung) auf und deuten darauf hin, dass einem Vermögenswert Sockel, Zubehör oder Verpackung zugeordnet werden kann, aber nicht zwingend muss. Darüber hinaus bestehen optionale One-to-Many Verbindungen zwischen der **assets** und den **customers** (Kunden), **exhibitions** (Ausstellungen) sowie **worksfamilies** (Modelfamilien). Die Collection **users** verwaltet Benutzer im System und hat keine Beziehung zu anderen Objekten. Die Collection **codices** ist für die Verwaltung von Codices von Leonardo da Vinci vorgesehen, die mit der Collection **pages** (Seite) eine One-to-Many Beziehung hat.

Nach der Gruppierung der Objekte wurden die Attribute für diese festgelegt, basierend auf den vom Projektteam Da Vinci bereitgestellten Daten. Die vollständige Visualisierung der Datenbankstruktur, einschließlich ihrer Attribute und Beziehungen, befindet sich Anhang. In diesem Abschnitt wird nur zentrale Collection `assets` detailliert erläutert, während die anderen Collections auf ähnliche Weise implementiert wurden.

Im Folgenden wird Collection `assets` erläutert. Die Abb.22 präsentiert das Dokument-schema der Collection.

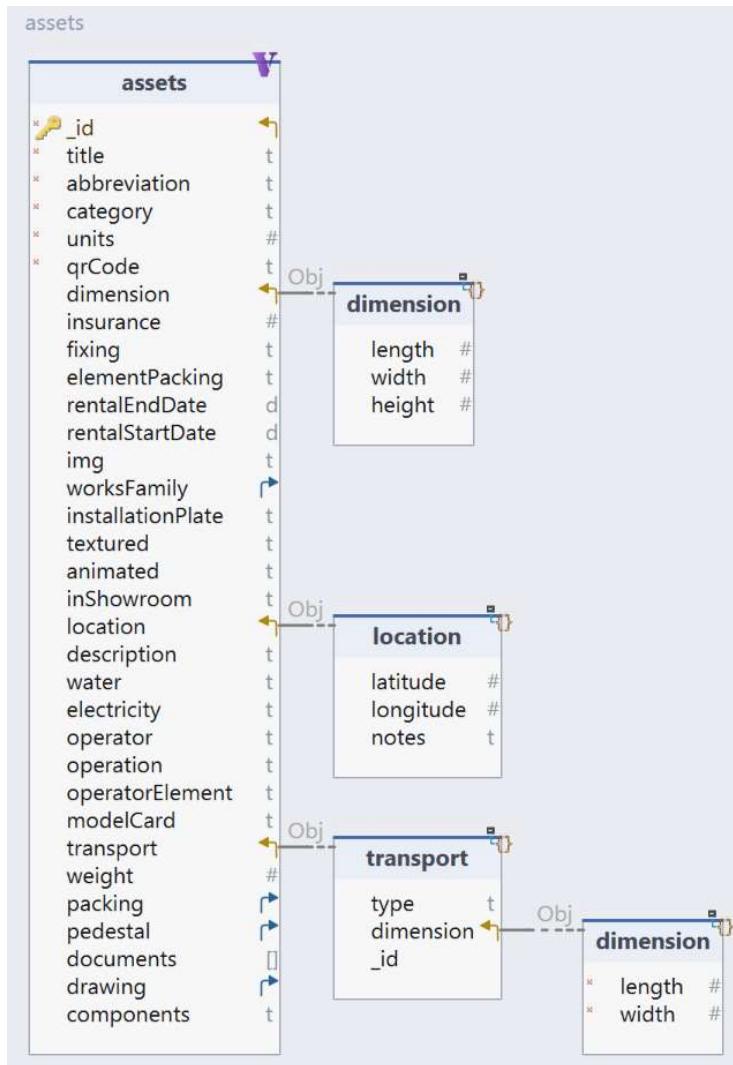


Abb. 24: Dokumentstruktur eines Vermögenswerts. Die gelben Pfeile zeigen auf eingebettete Daten, die blaue auf Referenzen.

Sie beinhaltet ein automatisch generiert Primärschlüssel `_id`, der das Document eindeutig in der Datenbank repräsentiert und für Referenzen dient. Die Attribute `title` (Bezeichnung), `abbreviation` (Kürzel) und `category` (Kategorie) sind obligatorisch beim Anlegen des Vermögenswerts im System. Die Kategorie des Vermögenswerts ermöglicht

es, verschiedene Arten von Vermögenswerten zu unterscheiden. Es wurden folgende Kategorie der Vermögenswerte festgelegt:

- Mechanisches Modell;
- Virtuelles Modell;
- Banner;
- Gemälde;
- Gestaltungselement;
- Gestaltungsobjekt;
- Informationstafel;
- Skizze Anatomie;
- Skizze Technik;
- Sonstige Skizze;

Durch die Verwendung des *category*-Attributs können Abfragen gezielt optimiert werden, ohne überflüssige Daten durch mehrere Sammlungen hinweg durchsuchen zu müssen.

In Weiterem wurden innerhalb des Documents wurden drei eingebettete Objekte angelegt: *dimension* (Abmessung), *location* (Standort) und *transport* (Transport). Die eingebetteten Objekte sind mit gelben Pfeilen markiert. Unter der *dimension* wurde Länge, Breite und Höhe des Werts definiert. Die *location* dient, um die GPS-Daten sowie Standortnotizen zu speichern. Das *transport* Objekt einbezieht wiederum ein eingebettetes Objekt, das ein *\_id*-Schlüssel-, type- und *dimension*-Attribute enthält. Dieser Schlüssel dient für eindeutige Indizierung der Transportinformationen innerhalb der Datenbank. Das Attribut *type* beschreibt den Transportart, etwa das *Europalette* oder *Sockelpalette*. Zudem wurden in der *dimension* die Abmessungen einer Palette festgehalten. Diese eingebetteten Datenstrukturen ermöglichen es, alle relevanten Informationen zu einem Vermögenswert innerhalb desselben Dokuments zu speichern und abzurufen, ohne dass auf andere Tabellen oder Sammlungen zugegriffen werden muss.

Zusätzlich zu den eingebetteten Feldern enthält die *assets* Collection Referenzen zu anderen Collections oder Dokumenten. Diese Referenzen sind im Diagramm durch blaue Pfeile gekennzeichnet und ermöglichen die Verknüpfung eines Vermögenswerts durch den *\_id*-Schlüssel, die detailliertere oder weiterführende Informationen

enthalten. Es besteht Referenzen zu *worksfamilies* (Modelfamilie), *pedestals* (Sockel), *accessories* (Accessuar), *packing* (Modellverpackung) und *assets*. Beim *assets* handelt es sich um das Referenzieren auf sich selbst, um eine Skizze eines mechanischen Modells zu hinterlegen.

Die Datenbank wurde in Node.js unter Verwendung des NPM-Pakets *Mongoose* erstellt. *Mongoose* ist ein Framework, das speziell für die objektorientierte Modellierung in MongoDB entwickelt wurde. Es basiert auf einem halbstarren Schema und beschleunigt die Entwicklung der Datenbank. Jedes Objekt besteht aus einem Schema. Aus dem Schema kann ein Modell definiert werden, welches der Struktur eines Document in MongoDB entspricht. Dabei ein Modell kann aus mehreren Schemas bestehen. Bei der Anlage eines Modells wird in der Datenbank eine Collection erstellt.

Zusammenfassend ermöglicht die Struktur der Datenbank eine effiziente Verwaltung und Abfrage von Vermögenswerten, indem sowohl eingebettete Datenstrukturen als auch Referenzen zu anderen Collections genutzt werden. Diese Kombination erlaubt es, alle relevanten Informationen zentral und konsistent zu speichern, was die Datenbankabfragen vereinfacht und beschleunigt.

Im nächsten Kapitel wird die API vorgestellt, die für die Kommunikation zwischen der Anwendung und der Datenbank zuständig ist.

## 6.5 REST-API

Im vorliegenden Kapitel folgt eine detaillierte Erläuterung zur Auswahl und Umsetzung einer geeigneten API. Gemäß den in Kapitel 5 gestellten Anforderungen muss die API eine Client-Server-Architektur unterstützen. Die Übertragung von Nachrichten muss effizient gestaltet werden, um eine hohe Leistungsfähigkeit zu gewährleisten. Da das Projekt voraussichtlich schnell wachsen wird und häufige Änderungen zu erwarten sind, muss die API eine skalierbare Lösung bieten, die sich leicht an eine wachsende Anwendung anpassen lässt. Darüber hinaus sind robuste Sicherheitsmechanismen erforderlich, um sicherzustellen, dass nur autorisierte Benutzer auf Daten zugreifen können. Angesichts der stark begrenzten Entwicklungszeit muss die API zudem eine schnelle Implementierung ermöglichen.

### 6.5.1 Vergleich API-Technologien

Für die oben genannten Anforderungen stehen verschiedene APIs zur Auswahl. Im Rahmen dieser Untersuchung werden drei Technologien berücksichtigt, die eine

Schnittstelle zwischen Frontend und Backend ermöglichen und in der modernen Webentwicklung weit verbreitet sind. Hierbei handelt es sich um Simple Object Access Protocol API (SOAP-API), GraphQL-API und Representational State Transfer API (REST-API). Der Vergleich basiert auf drei Kriterien: *Leistung*, *Skalierbarkeit*, *Sicherheit* und *Entwicklungszeit*. Jedes Kriterium wird auf einer Skala von 1 bis 5 bewertet, wobei 5 die beste Bewertung darstellt. Die Zusammenfassung der Bewertung wird in der folgenden Tabelle dargestellt und im Anschluss diskutiert:

Kriterium	SOAP-API	GraphQL-API	REST-API
Leistung	3	5	5
Skalierbarkeit	2	5	4
Sicherheit	5	4	4
Entwicklungszeit	3	3	5
Summe	13/20	17/20	18/20

Tab. 7: Vergleich API-Technologien

SOAP-Nachrichten sind aufgrund ihrer beträchtlichen Größe und Komplexität langsamer in der Übertragung und Verarbeitung. Im Gegensatz dazu werden bei GraphQL-API-Anfragen nur die benötigten Daten durch den Client angefordert, was die Übertragung beschleunigt. Auch REST-API-Nachrichten sind in der Regel klein und daher schnell übertragbar. In der Gesamtbewertung erhält SOAP-API eine Bewertung von 3, GraphQL-API und REST-API jeweils 5 Punkte.

SOAP-APIs zeigen sich zudem weniger flexibel und erfordern einen höheren Aufwand für die Skalierung, besonders bei zunehmender Anwendungskomplexität. GraphQL-API eignet sich gut für die Skalierung, da es effizient mit Daten umgeht und der Client nur die benötigten Informationen anfordert. REST-APIs sind ebenfalls skalierbar, benötigen jedoch eine sorgfältige Verwaltung von Endpunkten und eine Versionierung, besonders bei Änderungen in der Datenstruktur. Dies führt zu einer Gesamtbewertung von 2 für SOAP-API, 5 für GraphQL-API und 4 für REST-API.

In Bezug auf Sicherheit bieten SOAP-APIs umfangreiche Mechanismen wie Web Service Security, ein leistungsfähiges Sicherheitsprotokoll. Auch bei GraphQL-API und REST-APIs ist eine sichere Gestaltung möglich, erfordert jedoch zusätzliche Implementierungen für Authentifizierung und Autorisierung. Daher erhält SOAP-API hier die

höchste Bewertung, gefolgt von GraphQL-API und REST-API, was zu einem Verhältnis von 5 zu 4 zu 4 führt.

Die Komplexität von SOAP-API führt oft zu längeren Entwicklungszeiten, insbesondere aufgrund der detaillierten Beschreibung des Web-Services. Auch die Einarbeitung in GraphQL-API erfordert Zeit, insbesondere bei der Definition des Schemas. REST-APIs hingegen sind einfacher und schneller zu implementieren, da sie weniger formalen Strukturen folgen. Die resultierenden Bewertungen sind daher 3 für SOAP-API, 3 für GraphQL-API und 5 für REST-API.

Die Summe der Bewertungen zeigt, dass sowohl GraphQL-API als auch REST-API gut für dieses Projekt geeignet sind. REST-APIs bieten insgesamt die beste Kombination aus Leistung, Skalierbarkeit, Sicherheit und Entwicklungszeit für die diese Anwendung. Aufgrund der schnelleren Entwicklungszeit ist REST-API mit dem Punktestand 18 von 20 besser geeignet als GraphQL-API mit 17 zu 20.

### 6.5.2 Einführung

REST-APIs sind Schnittstellen, die auf den Prinzipien der REST-Architektur basieren. Diese Architektur bietet eine strukturierte und effiziente Methode zur Kommunikation zwischen Client und Server. Die wesentlichen Grundsätze der REST-Architektur lassen sich wie folgt zusammenfassen:

- **Einheitliche Schnittstelle:** Die Kommunikation zwischen Client und Server erfolgt in einem standardisierten Format, das eine konsistente Datenübertragung gewährleistet.
- **Zustandslosigkeit:** Jede Anfrage vom Client wird unabhängig und isoliert behandelt, ohne dass der Server den Zustand vorheriger Anfragen speichert. Dadurch können Anfragen in beliebiger Reihenfolge gesendet werden, ohne dass dies die Funktionsweise der API beeinträchtigt.
- **Schichtenmodell:** Die Architektur ermöglicht die Implementierung mehrerer Schichten, wobei mehrere Server zusammenarbeiten können, um die Anfragen des Clients zu bearbeiten. Der Client ist jedoch nicht in der Lage, die interne Struktur des Servers zu erkennen.
- **Cachefähigkeit:** Antworten des Servers können zwischengespeichert werden, was die Effizienz erhöht und die Reaktionszeit des Servers verringert.

Der Kommunikationsprozess beginnt, wenn der Client über die API eine Ressource vom Server anfordert. Jede Ressource wird dabei durch eine eindeutige Uniform Resource Locator (URL) identifiziert, die als Zieladresse für die Anfrage fungiert. Diese URL gibt nicht nur die Ressource an, sondern auch die Art der Aktion, die der Server mit der Ressource ausführen soll. Zu jeder URL wird eine HTTP-Methode hinzugefügt, die den gewünschten Vorgang beschreibt:

- GET-Methode: Diese Methode wird verwendet, um eine Darstellung der angegebenen Ressource anzufordern. Der Server stellt die angeforderten Daten zur Verfügung, wobei die Möglichkeit besteht, diese nach bestimmten Kriterien zu sortieren oder zu filtern.
- POST-Methode: Diese Methode dient dem Senden von Daten an den Server, um eine neue Instanz der Ressource zu erstellen oder eine bestehende Ressource zu aktualisieren.
- PUT-Methode: Diese Methode wird ausschließlich zur vollständigen Aktualisierung einer bestehenden Ressource auf dem Server verwendet.
- DELETE-Methode: Mit dieser Methode wird die angeforderte Ressource vom Server gelöscht.

Jede dieser Anfragen wird unter Berücksichtigung der HTTP-Header verarbeitet, die Metadaten wie das Datenformat oder Authentifizierungsinformationen enthalten. Darüber hinaus können in der URL-Syntax Parameter angegeben werden, die zusätzliche Anweisungen zur Verarbeitung der angeforderten Ressource enthalten.

REST-APIs zeichnen sich durch ihre Skalierbarkeit, Flexibilität und Unabhängigkeit aus. Flexibilität bedeutet hier die klare Trennung zwischen Client und Server, wodurch beide unabhängig voneinander entwickelt werden können. Diese Unabhängigkeit ermöglicht es zudem, Client- und Serveranwendungen in unterschiedlichen Programmiersprachen zu entwickeln, ohne das Design der API zu beeinflussen.

### 6.5.3 Express

Für die Entwicklung der REST-API wird das Webframework `express.js` genutzt. Express ist ein schnelles und minimalistisches Web-Framework für Node.js, welches die effiziente Erstellung von Webanwendungen und APIs ermöglicht. Express stellt

grundlegende Funktionen zur Verfügung, darunter Routing, Middleware und die Verarbeitung von HTTP-Anfragen. Zur Handhabung von HTTP-Anfragen und -Antworten stehen das Anfrageobjekt (req) und das Antwortobjekt (res) bereit. Das "req"-Objekt verfügt über Eigenschaften, die den Anfragestring, die Parameter, den Body sowie die HTTP-Header betreffen. Das res-Antwort-Objekt dient der Generierung einer HTTP-Antwort, welche anschließend an den Client zurückgegeben wird. Middleware in einer REST-API ist eine Funktion, die Zugriff auf das Request-Objekt, das Response-Objekt und die nächste Middleware-Funktion im Request-Response-Zyklus der Anwendung hat. Die Middleware hat Zugriff auf das Anforderungsobjekt und das Antwortobjekt und kann die Anforderung verarbeiten, bevor der Server eine Antwort sendet. Eine Express-basierte Anwendung besteht aus einer Reihe von Middleware-Funktionsaufrufen. Des Weiteren verfügt es über Methoden zum Zurücksenden der HTTP-Antwort, wobei verschiedene Elemente wie Inhalt, Statuscode, Kopfzeile und Ähnliches berücksichtigt werden können. Die genannten Objekte stellen die Grundlage für die Bearbeitung von HTTP-Anfragen und -Antworten in Node.js-Anwendungen mit Express dar. Sie ermöglichen Entwicklern den Zugriff auf Anforderungsinformationen, beispielsweise auf Header, Parameter und den Body-Inhalt, sowie die Definition der Antwort, die an den Client zurückgesendet werden soll. Dazu zählen unter anderem Statuscodes, JSON-Inhalt und weitere Elemente.

#### 6.5.4 Initialisierung des Servers

Die Implementierung der REST-API und der Datenbank erfolgt in der Node.js-Laufzeitumgebung. Node.js ist eine plattformübergreifende Open-Source-Laufzeitumgebung, die die Ausführung von JavaScript-Anwendungen auf verschiedenen Betriebssystemen ermöglicht. Sie bietet Schnittstellen zur Interaktion mit dem Betriebssystem, einschließlich Zugriff auf das Dateisystem, wodurch das Lesen und Schreiben von Dateien ermöglicht wird. Darüber hinaus stellt Node.js den NPM (Node Package Manager) bereit, um die Installation externer Pakete zu ermöglichen.

Zur Erstellung und Verwaltung der Verbindung zu MongoDB kommt das *mongoose*-Paket zum Einsatz. Für die einfache Handhabung von Dateien wird das NPM-Paket *multer* verwendet. Diese Bibliotheken bilden die Kernkomponenten der Backend-Entwicklung. Der Einsatz weiterer Bibliotheken für spezifische Anwendungsfälle wird im Verlauf der Implementierung detailliert erläutert. Im Folgenden wird Initialisierung der Server vorgenommen.

Die Initialisierung der Server umfasst folgende Schritte:

- Erstellung der Anwendungsinstanz
- Konfiguration der CORS- und JSON-Middleware
- Definition der Routen, um auf die Datenbank zu greifen
- Konfiguration der statischen Routen für die Speicherung von Dateien
- Herstellung der Verbindung mit der Datenbank
- Start des Servers

Im Folgenden folgt eine Erläuterung der einzelnen Schritte der Initialisierung. Durch den Aufruf der Funktion `express()` wird eine Instanz erzeugt, die anschließend zur Konfiguration des Servers verwendet wird. Danach wird die Middleware für die Zulassung von http-Anfragen und zur Konvertierung von JSON-Objekten konfiguriert. Cross-Origin Resource Sharing (CORS) ist ein Sicherheitsmechanismus, der es einem Server ermöglicht, festzulegen, von welchen anderen Ursprüngen ein Browser das Laden von Ressourcen erlauben darf. Hierzu sendet der Browser zuvor eine Anfrage an den Server, um zu prüfen, ob die Anfrage zugelassen wird.

Da die Anwendung im Localhost ausgeübt wird, sind derzeit keine spezifischen Einstellungen für den Serverzugriff erforderlich. Die JSON-Middleware konvertiert die JSON-Anfragen in JavaScript Variablen, um im Anschluss die Variablen zu verarbeiten. Anschließend werden die Routen definiert, um den Zugriff auf die Datenbank zu ermöglichen. Für die Speicherung von Bildern, PDF-Dateien und QR-Codes werden statische Routen definiert. Danach wird die Verbindung zu MongoDB hergestellt. Die Verbindung zu MongoDB wird mit Hilfe einer Instanz von `mongoose` realisiert, indem `mongoose` mittels MongoDB-Connection-URI konfiguriert wird. Die Connection-URI wird z.B. im *MongoDBCompass* festgelegt. Abschließend erfolgt der Start des Servers.

### 6.5.5 Authentifizierung und Autorisierung

Um die Sicherheitsanforderungen zu erfüllen, muss die API einen Authentifizierungs-Prozess abbilden. Die Authentifizierung ist ein entscheidender Sicherheitsaspekt zur Gewährleistung, dass nur berechtigte Benutzer oder Anwendungen bestimmte Operationen durchführen können. Es gibt drei übliche Authentifizierungsmethoden, die in einer REST-API üblicherweise zum Einsatz kommen:

- Basis-Authentifizierung: Bei der Basis-Authentifizierung wird der Benutzername und das Passwort in einem HTTP-Header mit jeder Anfrage gesendet.

- Bearer-Authentifizierung: Diese Methode setzt voraus, dass jede Anfrage an den Server einen gültigen Authentifizierung-Token nachweisen muss, um den Zugang zur Ressource zu erhalten. Das Authentifizierung-Token ist eine kryptische Zeichenfolge, die vom Server als Antwort auf eine Login-Anfrage generiert wird.
- API-Schlüssel: In diesem Ansatz wird einem Client ein eindeutiger API-Schlüssel zugewiesen, mit dem sich der Client bei der API verifiziert. API-Schlüssel sind weniger sicher, weil der Client den Schlüssel übertragen muss, was ihn gegenüber Netzwerkangriffen anfällig macht.
- OAuth: OAuth wird vom Framework OAuth2 bereitgestellt, das Passwörter und Tokens kombiniert, um einen sicheren Anmeldezugriff auf alle Systeme zu gewährleisten.

In diesem Projekt wird die API per Bearer-Authentifizierung umgesetzt. Das Verfahren wird in der OpenAPI 3.0 Spezifikation genauer definiert [29].

Für den Token wird das JSON Web Token (JWT) eingesetzt. Die Erstellung des JWT erfolgt in Übereinstimmung mit den Vorgaben des Industriestandards RFC 7519 [30]. Der Standard legt fest, aus welchen Bestandteilen sich der Token zusammensetzt. Die Erstellung des JSON Web Tokens erfolgt mit Hilfe des NPM-Paket *jsonwebtoken* und unter Zuhilfenahme des Algorithmus HS256, in dem die Benutzerdaten sowie eine vordefinierte Zeichenkette einbezogen werden. Im Rahmen der Anmeldung der Benutzer erfolgt eine Speicherung des Tokens in der Datenbank. Die Gültigkeitsdauer des Tokens beläuft sich auf 24 Stunden.

Die Abb. zeigt ein Sequenzdiagramm und veranschaulicht den Authentifizierungsprozess der REST-API, indem es den Ablauf der Kommunikation zwischen dem Frontend, der API und dem Backend beschreibt.

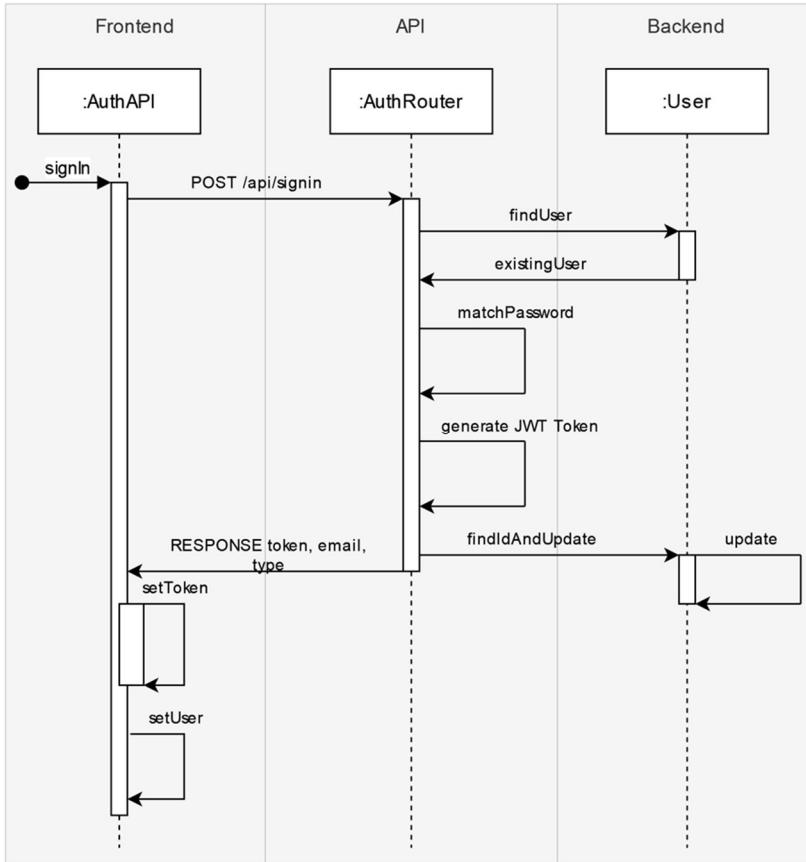


Abb. 25: Login Sequenzdiagramm

Der Prozess beginnt, wenn ein Benutzer im Frontend versucht, sich über die Methode `signIn` anzumelden. Diese Anmeldung löst eine POST-Anfrage an den Endpunkt `/api/signin` der API aus.

Der `AuthRouter` der API empfängt diese Anfrage und startet den Authentifizierungsprozess, indem er zunächst die Methode `findUser` im Backend auruft, um nach einem entsprechenden Benutzer in der Datenbank zu suchen. Das Backend überprüft die Datenbank und gibt den gefundenen Benutzer zurück an den `AuthRouter`.

Anschließend überprüft der `AuthRouter` das eingegebene Passwort des Benutzers, um sicherzustellen, dass es korrekt ist. Ist das Passwort korrekt, generiert der `AuthRouter` das `JWT`, das zur Authentifizierung des Benutzers bei zukünftigen API-Anfragen verwendet wird.

Der `AuthRouter` aktualisiert dann den Benutzerdatensatz in der Datenbank, um beispielsweise das Datum der letzten Anmeldung zu speichern. Diese Aktualisierung wird vom Backend bestätigt.

Nachdem alle Überprüfungen und Aktualisierungen erfolgreich durchgeführt wurden, sendet die API eine Antwort an das Frontend. Diese Antwort enthält das JWT, die E-Mail-Adresse des Benutzers und den Benutzertyp. Im Frontend wird der Token gespeichert, ebenso wie die Benutzerdaten, wodurch der Benutzer als authentifiziert gilt und auf geschützte Ressourcen der Anwendung zugreifen kann.

Der Registrierungsprozess wird durch den Administrator initiiert, wobei die Rolle des Benutzers innerhalb der Systemstruktur definiert wird. Die Authentifizierung erfolgt mittels einer E-Mail-Adresse sowie eines Passworts. Dabei ist die Wahl der E-Mail-Adresse und die Wahl des Benutzertyps als obligatorisch. Nach erfolgreicher Registrierung ist jeder Benutzer in der Lage, sein Passwort neu anzulegen. Die Speicherung der Passwörter erfolgt in kryptografischer Weise innerhalb der Datenbank unter Zuhilfenahme des NPM-Moduls *bcrypt*. Die kryptografische Hashfunktion *bcrypt* gewährleistet eine sichere Speicherung der Daten.

Um die Anforderungen /B03/, /B04/ und /B05/ bezüglich der Benutzerzugriffsrechte zu erfüllen, wurden spezielle Middleware-Funktionen implementiert. Eine dieser Middleware prüft, ob der aktuelle Benutzer als Admin angemeldet ist, während eine andere sicherstellt, dass der Benutzer entweder als Admin oder WMA authentifiziert ist. Diese Middleware ermöglichen einen differenzierten Zugriff auf die Ressourcen.

### 6.5.6 Routing

Das Routing, im Sinne der API, stellt Benutzern des Frontends eine Struktur von sog. Endpunkten zur Verfügung. Der Aufruf eines Endpunkts führt zu einer konkreten Methode und löst somit eine Funktionalität der API aus. Diese Funktionalität kann wiederum auf Daten der Datenbank zugreifen, ähnlich wie im vorigen Kapitel für die Authentifizierung und Autorisierung dargestellt. In diesem Abschnitt werden die im System vorhandenen Endpunkte dargestellt. Damit das Frontend eine Anfrage an einen Endpunkt senden kann, werden in einer REST-API folgende Methoden für Anfragen definiert:

- Die GET-Methode erlaubt allen autorisierten Clients, die Ressource zu lesen.
- Die Methoden POST und PUT sind ausschließlich den Administratoren und WMA-Nutzern vorbehalten, um Ressourcen anzufordern oder Funktionen im Backend auszulösen.

- Die DELETE-Methode gewährt ausschließlich dem Administrator Zugriff auf die Ressourcen und erlaubt die Löschung von Daten aus der Datenbank.

Die REST-API dient der Verwaltung von Benutzern, Vermögenswerten, Kunden, Ausstellungen, Modellfamilien und Codices. Obwohl die Verwaltung der Codices als optionales Kriterium vorgesehen ist und aufgrund der begrenzten Entwicklungszeit im Frontend nicht umgesetzt wird, bietet das System dennoch eine Schnittstelle für die Verwaltung der Codices und deren Seiten. Diese API-Integration stellt sicher, dass die Verwaltung der Codices zu einem späteren Zeitpunkt im Frontend implementiert werden kann, was auch zu einer schnelleren und effizienteren Weiterentwicklung des Systems beitragen wird.

Im Anhang befindet sich eine Übersicht aller Routen, die von der API bereitgestellt werden, zusammen mit den erforderlichen Zugriffsrechten.

Eine grafische Darstellung aller Endpunkte kann der nachfolgenden Tabelle entnommen werden. Im Folgenden erfolgt eine Beschreibung der Verwendungszwecke der verschiedenen Endpunkte unter Berücksichtigung der jeweiligen HTTP-Methoden. Dabei wird zugleich die dahinterliegende Businesslogik erörtert.

### 6.5.7 Fehlerbehandlung

Anfragen an die API können aus diversen Gründen scheitern, bspw. bei mangelnder Verbindung vom Client zum Server oder auch wenn einem Benutzer die entsprechende Berechtigung fehlt. Um zu prüfen, ob eine Anfrage erfolgreich war, wurden Statuscodes implementiert. Statuscodes sind integraler Bestandteil der Fehlerbehandlung in REST-APIs und ermöglichen die Ursachen bei gescheiterten Anfragen zu erörtern.

Die Fehlerbehandlung in REST-APIs umfasst die Beantwortung von Anfragen mit entsprechenden HTTP-Statuscodes und Meldungen, welche den Erfolg oder Misserfolg einer Operation anzeigen. Der HTTP-Statuscode findet sich in der Open-API-Spezifikation *RFC 9110* als Empfehlung und wird in der Anwendung zur Fehlererkennung eingesetzt [31]. In der Spezifikation lässt sich HTTP-Statuscodes in fünf Hauptklassen unterteilen:

- 100 bis 199: Beschreiben informationelle Antworten
- 200 bis 299: Beschreiben erfolgreiche Anfragen
- 300 bis 399: Sind als Umleitungsnachrichten definiert

- 400 bis 499: Beinhalten Informationen über Client-Fehlerantworten

Die Verarbeitung der Serverantworten erfolgt mithilfe des *res-Antwortobjekts*. Die von Express bereitgestellte Methode `res.status()` wird verwendet, um den entsprechenden Statuscode basierend auf der erwarteten Antwort festzulegen. Somit wird sichergestellt, dass der Client eine klare und präzise Rückmeldung erhält.

Die folgende Tabelle stellt die für dieses Projekt implementierten Statuscodes der REST-API dar. Der Status ist hierfür grob in erfolgreiche Anfragen und fehlerhafte Anfrage generiert. Diesem Status sind die jeweiligen Codes zugeordnet und für jeden Statuscode ist eine Beschreibung der Bedeutung zu finden.

Status der Anfrage	Statuscode	Bedeutung
Erfolgreiche Anfrage	200	Der Statuscode wird für erfolgreiche GET-, PUT-, PATCH-Anfragen zur Aktualisierung von Daten verwendet.
	204	Der Statuscode 204 findet bei Delete-Anfragen Anwendung. Dies bedeutet, dass der Server die Anforderungen erfolgreich erfüllt hat und keine weiteren Inhalte in der Antwort zu senden sind.
Fehlerhafte Anfrage	400	Der Server ist in diesem Fall nicht in der Lage, die Anfrage zu bearbeiten.
	401	Es liegen keine validen Authentifizierungsdaten für die angefragte Ressource vor.
	404	Es liegen keine validen Authentifizierungsdaten für die angefragte Ressource vor. Die Anfrage kann nicht bearbeitet werden, da der Ursprungsserver keine aktuelle Darstellung für die Zielressource gefunden hat.
	500	Der Server trifft auf eine unerwartete Bedingung, die einer Erfüllung der Anfrage entgegensteht.

Tab. 8: Statuscodes der implementierten API

### 6.5.8 Dateimanagement

Im Rahmen der Implementierung des Dateimanagements wird das NPM-Paket *Multer* installiert. *Multer* stellt eine Node.js-Middleware-Bibliothek dar, welche für die Speicherung der Dateien verantwortlich ist und HTTP-Anfragen mit dem Content-Type *multipart/form-data* unterstützt. Der Datentyp HTTP-Multipart ermöglicht die Codierung von Informationen als eine Reihe von Teilen in einer Nachricht.

Im Rahmen der Speicherung der Dokumente wird eine Middleware implementiert, welche die Festlegung des Dateinamens sowie des Speicherorts übernimmt. Die maximale Größe einer Datei ist auf 5 MB beschränkt (Anforderung /PL30/). Express verwendet die Funktion *use*, um Dateien statisch zu speichern. Die Erstellung statischer Ordner erfolgt beispielsweise durch die Anweisung `app.use("/drawings", express.static("drawings"))`. Die vorliegende Konfiguration definiert die Bereitstellung von Dateien aus dem Verzeichnis *drawings* bei Anfragen an */drawings*. Der Zugriff auf die statische Route kann durch das Hinzufügen einer weiteren Middleware geschützt werden. Für das Verzeichnis */documents*, in dem die PDF-Dateien angelegt sind, wurde ein geschützter Zugriff umgesetzt. Auf die Dateien in diesem Verzeichnis können nur autorisierte Benutzer zugreifen.

## 7 Evaluierung

Im Anschluss an die Implementierung wurde eine ausführliche Evaluierung der Web-Anwendung durchgeführt. Der Evaluierungsprozess umfasst drei Phasen: Vorbereitung der Testumgebung, Akzeptanztest und Performance-Test. Ziel der Evaluierung ist, die Anwendung auf ihre Funktionsfähigkeit, Benutzerfreundlichkeit und die Einhaltung der im Pflichtenheft definierten Anforderungen zu validieren.

### 7.1 Technische Spezifikationen der Testumgebung

Im Rahmen der Qualitätssicherung wird eine lokale Testumgebung bereitgestellt, welche die Evaluierung der Web-Anwendung ermöglicht. Dazu wird in der Entwicklungs-Umgebung eine optimierte Version der Anwendung für den lokalen Webserver erstellt. Dies umfasst die Konvertierung des Dart-Codes in JavaScript sowie die Erstellung aller erforderlichen Dateien mittels Flutter, welche anschließend auf einem Webserver bereitgestellt werden. In der Folge kann die Anwendung von diversen Geräten innerhalb des gleichen Netzwerkes vom Webserver abgerufen und evaluiert werden.

Der Webserver wurde auf einem Windows 10 System mit einem x64-Prozessor, 32 GB Arbeitsspeicher und einer 8-Core-CPU installiert. Auf diesem Server wurde das Flutter SDK in Version 3.3.3 sowie Node.js in Version 18.12.2 installiert, um eine stabile und leistungsfähige Testumgebung zu gewährleisten.

## 7.2 Akzeptanztest

Das Projektteam Da Vinci nutzt die Anwendung und hat den Akzeptanztest durchführen in Form von *Black-Box-Tests*, durchführen können. Der Blackbox-Test ist eine Methode zur Prüfung der Systemfunktionen aus der Perspektive des Benutzers, ohne die internen Strukturen oder den Code zu betrachten. Ziel des Akzeptanztests war es, potenzielle Fehler aufzudecken und sicherzustellen, dass die Software den Erwartungen der Projektmitglieder genügt. Während der Tests haben die Projektmitglieder das System genauer kennengelernt. Zu Beginn der Akzeptanztest wurde die Anwendung erfolgreich auf einem Webserver bereitgestellt. Die Tests wurden auf einem Desktop im Browser Chrome und Microsoft Edge durchgeführt. Die Testfälle wurden auf Basis der im Pflichtenheft definierten Produktfunktionen erstellt. Während die Benutzer unter Anweisungen die Testfälle durchführten, wurden die Testfälle mittels folgender Vorlage vorbereitet und protokolliert:

ID	TS-01
Produktfunktion	/PF10/ Anmeldung im System
Benutzergruppe	Admin, WMA, SMA
URL	http://localhost:8080/api/signin
Fenstergröße	Klein, mittel, groß
Browser	Chrome, Firefox, Safari, Microsoft Edge
Erwartetes Ergebnis	Der Benutzer wird erfolgreich im System angemeldet und auf die Startseite weitergeleitet.
Tatsächliches Ergebnis	

Tab. 9: Vorlage eines Testfall

Im Rahmen des Akzeptanztests wurden, mithilfe der bereitgestellten Anweisungen, die folgenden Produktfunktionen überprüft: /PF10/, /PF20/, /PF50/, /PF70/, /PF80/, /PF120/. Die vollständigen Testfälle sind im Anhang dokumentiert.

Das Layout der Anwendung entsprach den Erwartungen der Benutzer. Die zwei unterschiedlichen Menüs für Desktop- und Mobilgeräte haben sich positiv auf das Benutzererlebnis ausgewirkt. Die Dialogfenster wurden als gut strukturiert empfunden und enthielten hilfreiche Hinweise im Falle falscher Datenformate. Die Zugriffsrechte im System waren klar definiert und wurden korrekt umgesetzt. Die Benutzeroberfläche, mit großen, gut lesbaren Texten wurde als ansprechend bewertet. Die Farbpalette und das helle Design der Anwendung wurden positiv aufgenommen. Das Layout der Anwendung passt sich dem jeweiligen Gerät gut an. Design- und Darstellungsfehler wurden während der Testpause behoben und anschließend erneut überprüft. Dies führte jedoch zu Einschränkungen bei der Nutzung des Systems. So konnten beispielsweise das Scannen von QR-Codes und das Auslesen von GPS-Daten aufgrund unsicherer Verbindungen nicht getestet werden, da diese Funktionen vom Browser blockiert wurden. Weitere Produktfunktionen werden im Folgenden näher erläutert.

### 7.3 Performance-Test

Für die Durchführung von Performance-Tests wurden Verarbeitungszeiten und -größen gemessen. Die Dateneingaben wurden in akzeptabler Zeit verarbeitet. Das System erfüllt die Anforderung */PL10/ Allgemeine Systemreaktionszeit* und bietet eine zufriedenstellende Benutzererfahrung. Die Anforderung */PL20/ Zeit zur Ausführung der Datenbankoperationen* wurde eingehalten; ein Auszug der entsprechenden Daten ist im Anhang, in der Abb. 30, aufgeführt und die Operation hat dabei nie 50ms überschritten. Gemäß */PL30/ Datentransfervolumen* wurde getestet, dass das System Dateien bis zu einer Größe von 5 MB speichern kann. Die Web-Anwendung funktioniert zuverlässig, ist benutzerfreundlich, effizient und leicht anpassbar.

Insgesamt erfüllt das System die Qualitätsanforderungen des Pflichtenhefts. Die Benutzeroberfläche, einschließlich der Anforderungen */B01/ bis /B06/*, wurden von Benutzern akzeptiert. Auch die nicht-funktionalen Anforderungen wurden erfüllt und vom Projektteam abgenommen.

## 8 Fazit

In der vorliegenden Arbeit wurde die Entwicklung und Implementierung eines plattformübergreifenden mobilen Asset-Management-Systems für das „Da Vinci“-Projekt an der

Hochschule Bielefeld beschrieben. Das Ziel dieses Projekts war, eine digitale Lösung zur Verwaltung von Kunstwerken und deren Zubehör zu schaffen. Und wurde durch die Einführung eines flexiblen, skalierbaren und benutzerfreundlichen Systems realisiert, das sowohl auf mobilen Geräten als auch auf Desktop-PCs funktioniert. Abschließend wurden die im Pflichtenheft definierten Funktionen durch einen Akzeptanztest abgenommen.

## 8.1 Limitierungen

Die App wurde auf verschiedenen Geräten und von Endnutzern hinsichtlich der Anforderungen geprüft. Allerdings war der Zeitrahmen für die Entwicklung bereits sehr strikt gewählt. Eine intensive Prüfung des Backends und der REST-API war im gewählten Zeitrahmen nicht mehr möglich.

Wie im Pflichtenheft bereits abgegrenzt wurde, ist im System keine Funktionalität für die Nachverfolgbarkeit von Ereignissen, bspw. im Lieferprozess, implementiert worden. Ein funktionierender QR-Code Scanner und das Ermitteln der Anforderung stellen dennoch vorbereitende Maßnahmen für zukünftige Arbeiten.

## 8.2 Ausblick

Die in der Einleitung erwähnten Technologien bieten viele Möglichkeiten für die Weiterentwicklung des Systems. Beispielsweise können NFTs verwendet werden, um das Besitztum von Kunstwerken darzustellen [1] und eine Historie von Ausleihen zu etablieren. Durch Verwendung von künstlicher Intelligenz könnten die Lieferketten optimiert werden und Vermögenswerte könnten bei Lieferprozessen mittels Tracking von Ereignissen per GS 1 Link [4] informativer gesteuert werden. Die umgesetzte Arbeit bietet eine solide Grundlage für weitere Forschungen und Entwicklungen.

## Literaturverzeichnis

- [1] S. Casale-Brunet, P. Ribeca, P. Doyle und M. Mattavelli, „Networks of Ethereum Non-Fungible Tokens: A graph-based analysis of the ERC-721 ecosystem,“ *IEEE International Conference on Blockchain*, 2021.
- [2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville und Y. Bengio, „Generative adversarial networks,“ *Communications of the ACM*, 2020.
- [3] Y. Tolcha, A. Kassahun, T. Montanaro, D. Conzon, G. Schwering, J. Maselyne und D. Kim, „Towards Interoperability of Entity-Based and Event-Based IoT Platforms: The Case of NGSI and EPCIS Standards,“ *IEEE Access*, Nr. 9, 2021.
- [4] „GS1 Germany,“ [Online]. Available: <https://www.gs1-germany.de/standards/daten austausch/gs1-digital-link/>. [Zugriff am 25 08 2024].
- [5] A. Biørn-Hansen, T.-M. Grønli und G. Ghinea, „A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development,“ *ACM Computing Surveys*, 2018.
- [6] N. Koram und R. Garg, „Review on Mobile App Development: Tools and Techniques,“ *IEEE World Conference on Applied Intelligence and Computing*, 2023.
- [7] „IEEE Standard Glossary of Software Engineering Terminology,“ *IEEE Std 610.12-1990*, pp. 1-84, 1990.
- [8] D. Meiller, App-Entwicklung mit Dart und Flutter 2: Eine umfassende Einführung, Berlin, Boston: De Gruyter Oldenbourg, 2021.
- [9] „Einführung in JSON,“ [Online]. Available: <https://www.json.org/json-de.html>. [Zugriff am 25 08 2024].
- [10] „RFC 9114,“ [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc9114>. [Zugriff am 25 08 2024].
- [11] „HTML Standard,“ [Online]. Available: <https://html.spec.whatwg.org/multipage/>. [Zugriff am 25 08 2024].

- [12] P. Suresh, J. V. Daniel, V. Parthasarathy und R. H. Aswathy, „A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment,“ *International Conference on Science Engineering and Management Research (ICSEMR)*, pp. 1-8, 2014.
- [13] J. N. Wijaya und Y. A. Kusumawati, „User Interface Design of Seminar and Workshop Information Mobile Apps for Self-Development,“ *International Conference on Informatics, Multimedia, Cyber and Information System*, 2022.
- [14] „Material Design,“ [Online]. Available: <https://m3.material.io/>. [Zugriff am 25 08 2024].
- [15] „Human Interface Guidelines,“ [Online]. Available: <https://developer.apple.com/design/human-interface-guidelines>. [Zugriff am 25 08 2024].
- [16] W. Oliveira, B. Moraes, F. Castor und J. P. Fernandes, „Analyzing the Resource Usage Overhead of Mobile App Development Frameworks,“ *International Conference on Evaluation and Assessment in Software Engineering*, 2023.
- [17] M. Mahendra und B. Anggorojati, „Evaluating the performance of Android based Cross-Platform App Development Frameworks,“ *International Conference on Communication and Information Processing*, 2021.
- [18] R. Francesc, C. Gravino, M. Risi, G. Scanniello und G. Tortora, „Mobile app development and management: results from a qualitative investigation,“ *International Conference on Mobile Software Engineering and Systems*, 2017.
- [19] H. Balzert, Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering, Heidelberg: Spektrum Akademischer Verlag, 2009.
- [20] „VS Code,“ [Online]. Available: <https://code.visualstudio.com/download>. [Zugriff am 25 08 2024].
- [21] „Flutter Documentation,“ [Online]. Available: <https://docs.flutter.dev/>. [Zugriff am 25 08 2024].
- [22] „Node JS Package Manager,“ [Online]. Available: <https://nodejs.org/en/download/package-manager>. [Zugriff am 08 25 2024].

- [23] „MongoDB,“ [Online]. Available: <https://www.mongodb.com/try/download/enterprise>. [Zugriff am 25 08 2024].
- [24] „VS Code Terminal Basics,“ [Online]. Available: <https://code.visualstudio.com/docs/terminal/basics>. [Zugriff am 25 08 2024].
- [25] „React Native Docs,“ [Online]. Available: <https://reactnative.dev/docs/getting-started>. [Zugriff am 25 08 2024].
- [26] „SKIA,“ [Online]. Available: <https://skia.org/docs/>. [Zugriff am 25 08 2024].
- [27] „RFC 4627,“ [Online]. Available: <https://www.ietf.org/rfc/rfc4627.txt>. [Zugriff am 25 08 2024].
- [28] M. Kaufmann und A. Meier, SQL- & NoSQL-Datenbanken, Heidelberg: Springer Vieweg Berlin, 2023.
- [29] „OpenAPI Specification,“ [Online]. Available: <https://swagger.io/specification/>. [Zugriff am 25 08 2024].
- [30] „RFC 7419,“ [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc7519>. [Zugriff am 25 08 2024].
- [31] „RFC 9110,“ [Online]. Available: <https://www.rfc-editor.org/rfc/rfc9110.html>. [Zugriff am 25 08 2024].
- [32] „Statista,“ [Online]. Available: <https://www.statista.com/statistics/869224/world-wide-software-developer-working-hours/>. [Zugriff am 08 25 2024].

# Anhang

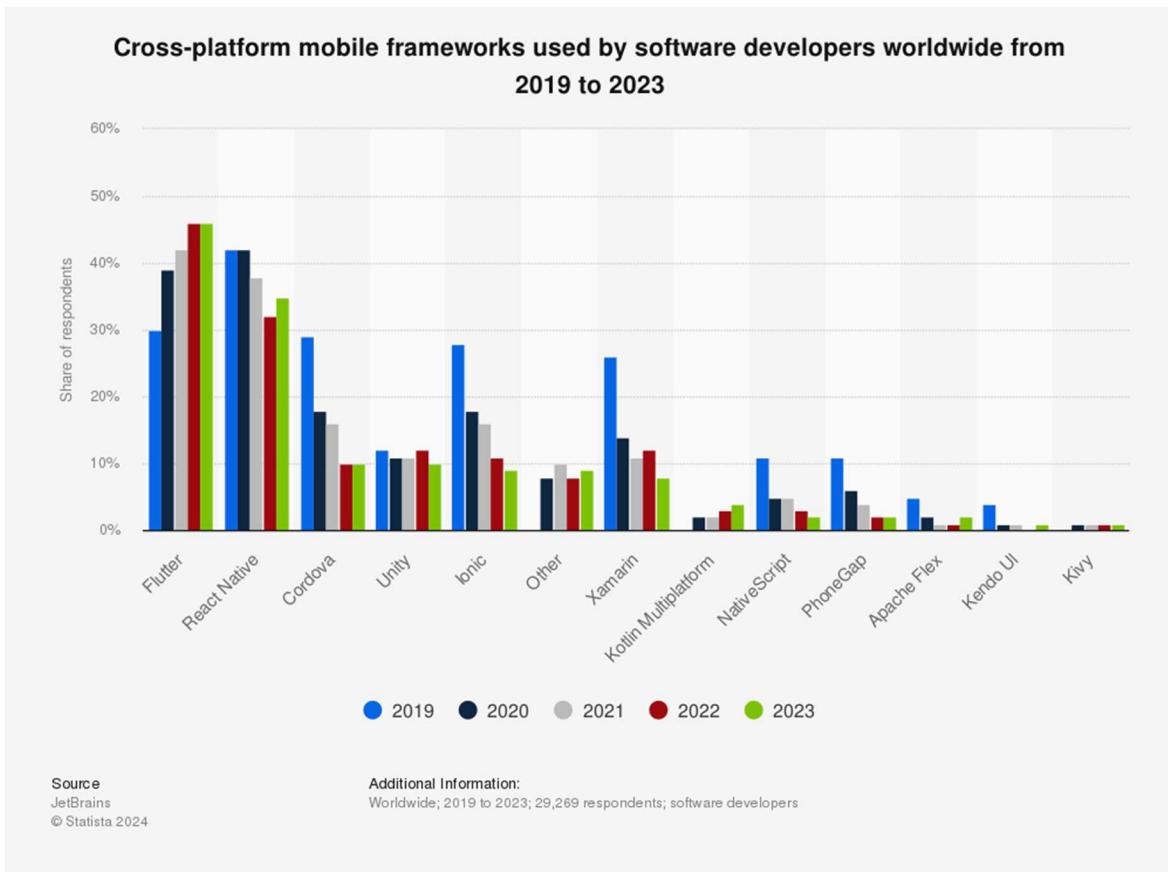
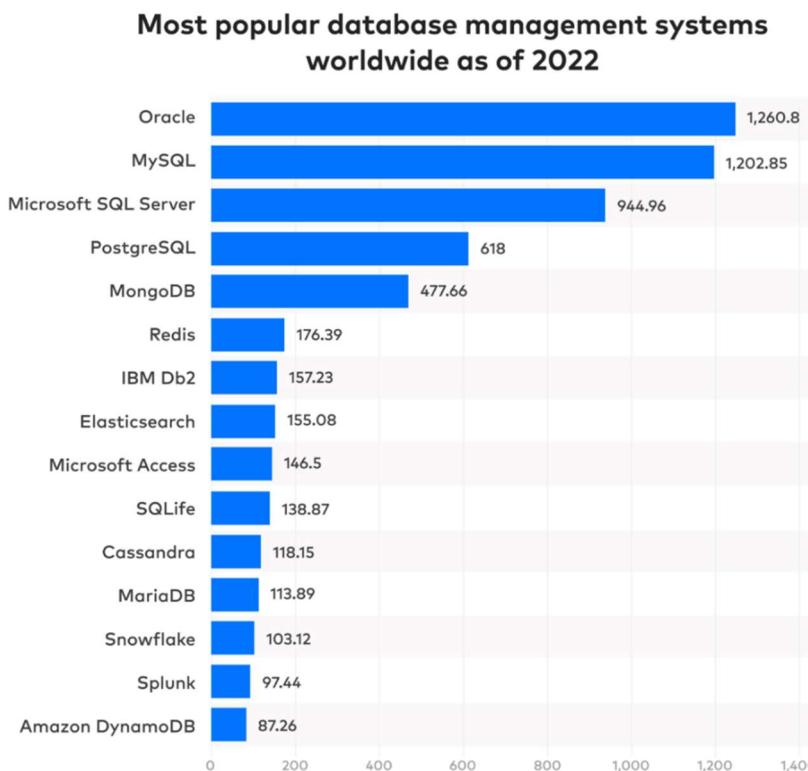


Abb. 26: Von Entwicklern weltweit verwendete Cross-Platform mobile Frameworks 2019-2023 [32]



Source: Statista

Abb. 27: Verteilung der beliebtesten Datenbankmanagementsysteme weltweit im Jahr 2022 [32].



Abb. 28: Anmeldefenster. Mobile Ansicht

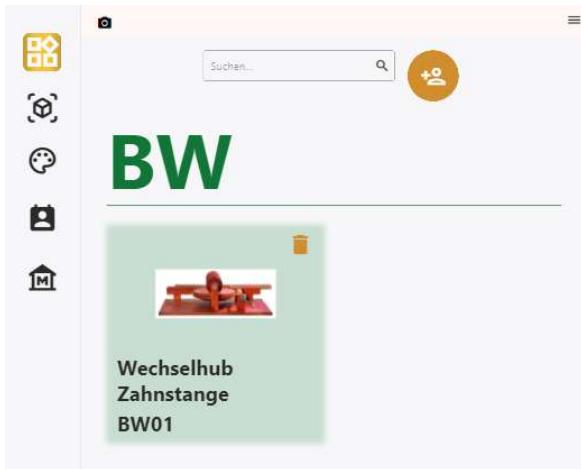


Abb. 29: Hauptfensteransicht bei Tablet- und Desktopansicht.

⌚ drawing?page=1&limit...	200	xhr	browser_client.dart:101	1.3 kB	12 ms
⌚ 66c352de98be2d7de9a...	200	xhr	browser_client.dart:101	1.1 kB	31 ms
⌚ 66c352de98be2d7de9a...	200	xhr	browser_client.dart:101	1.8 kB	12 ms

Abb. 30: Zeit der Ausführung der Datenbankoperationen. Die erste Spalte steht für "abrufen", die zweite für "speichern", die dritte für "aktualisieren".

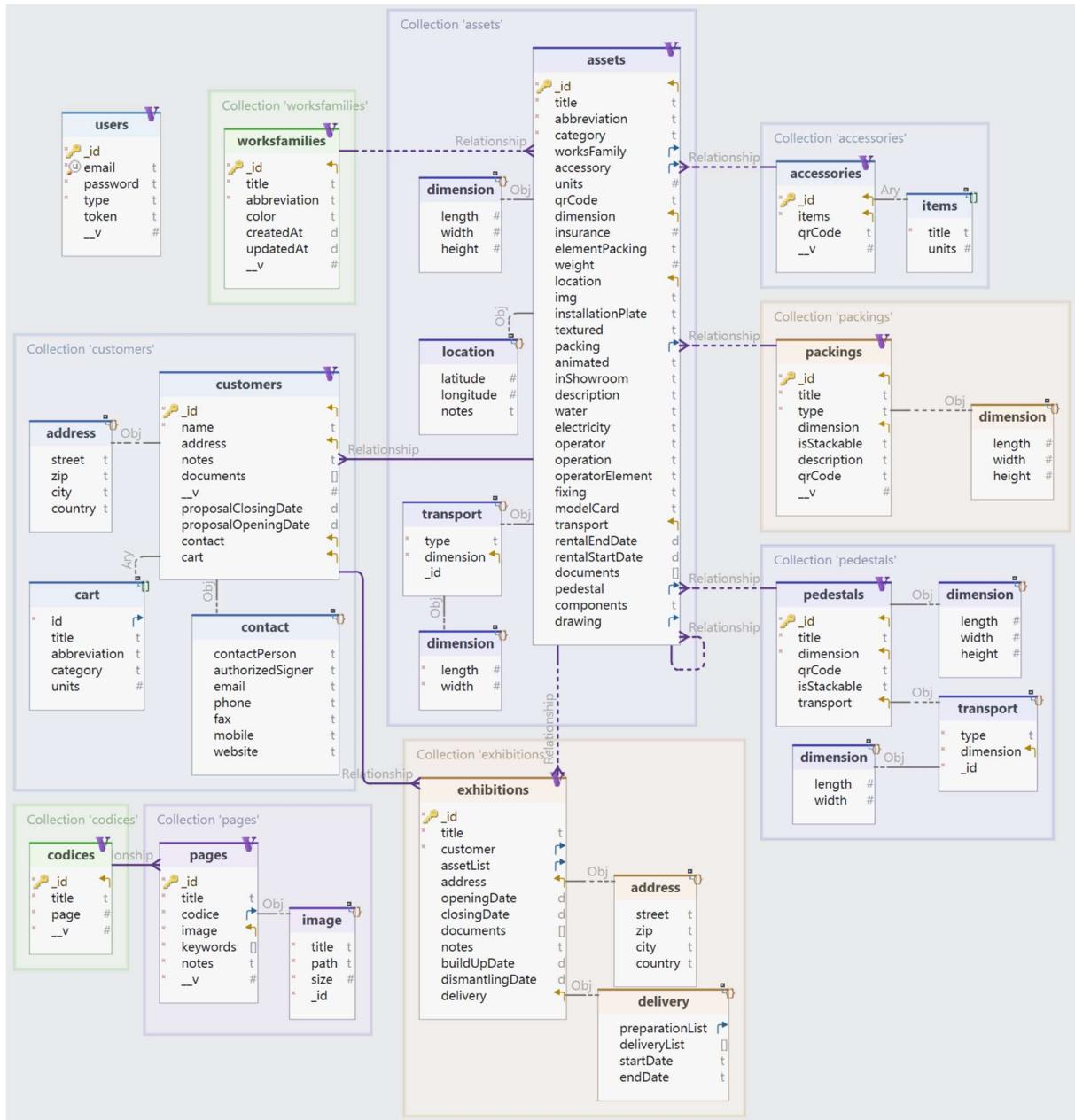


Abb. 31: Entity-Relationship-Modell des Systems.

HTTP-Methode	Beschreibung	Status und Antwort
/api/signup		
POST	Registriert einen neuen Benutzer	200: Benutzer erfolgreich registriert 400: Konto existiert bereits 500: Serverfehler
/api/signin		

POST	Meldet einen Benutzer an und gibt ein JWT zurück	200: JWT und Benutzerdaten 400: Benutzer nicht gefunden 403: Passwort falsch
<b>/api/reset-passwor</b>		
POST	Setzt das Passwort für den aktuellen Benutzer zurück	200: Passwort erfolgreich geändert 403: Serverfehler
<b>/api/logout</b>		
POST	Meldet den Benutzer ab und entfernt das Token	200: Erfolgreich ausgeloggt 500: Serverfehler
<b>/api/user/:id</b>		
DELETE	Löscht einen Benutzer	204: Benutzer gelöscht 500: Serverfehler
<b>/api/users</b>		
GET	Gibt eine Liste aller Benutzer zurück	200: Benutzerliste 500: Serverfehler
<b>api/work-families</b>		
GET	Gibt eine Liste aller vorhandenen Modellfamilien zurück.	200: Liste der Modellfamilien 500: Serverfehler
POST	Erstellt eine neue Modellfamilie.	200: Werksfamilie erfolgreich erstellt 400: Familie existiert bereits 500: Serverfehler
DELETE	Löscht eine Modellfamilie basierend auf der ID.	204: Werksfamilie erfolgreich gelöscht 500: Serverfehler
<b>api/customers</b>		
GET	Gibt eine Liste aller vorhandenen Kunden zurück.	200: Liste der Kunden 500: Serverfehler
POST	Fügt einen neuen Kunden hinzu.	200: Kunde erfolgreich hinzugefügt 400: Kunde existiert bereits 500: Serverfehler

PUT	Aktualisiert die Daten eines bestehenden Kunden.	200: Kunde erfolgreich aktualisiert 404: Kunde nicht gefunden 500: Serverfehler
DELETE	Löscht einen bestehenden Kunden, wenn keine Ausstellung mit ihm verbunden ist.	204: Kunde erfolgreich gelöscht 400: Kunde hat eine Ausstellung 500: Serverfehler
<i>api/customer/:id</i>		
GET	Gibt die Daten eines bestimmten Kunden zurück.	200: Kundendate 500: Serverfehler
<i>/add-day-to-proposal</i>		
POST	Fügt einem bestehenden Kunden Angebotsdatum hinzu.	200: Angebotsdaten erfolgreich hinzugefügt 404: Kunde nicht gefunden 500: Serverfehler
<i>api/customers/:id/add-to-proposal</i>		
POST	Fügt einem bestehenden Kunden ein Angebot hinzu.	200: Angebot erfolgreich hinzugefügt 404: Kunde nicht gefunden 500: Serverfehler
<i>api/customers/:id/document</i>		
POST	Aktualisiert die Dokumente eines bestehenden Kunden.	200: Dokument erfolgreich hinzugefügt 404: Kunde nicht gefunden 500: Serverfehler
DELETE	Löscht ein Dokument eines bestehenden Kunden.	200: Dokument erfolgreich gelöscht 404: Kunde nicht gefunden 500: Serverfehler
<i>api/exhibitions</i>		

GET	Gibt eine Liste aller Ausstellungen zurück.	<i>200: Liste der Ausstellungen</i> <i>500: Serverfehler</i>
POST	Fügt eine neue Ausstellung hinzu.	<i>200: Ausstellung erfolgreich erstellt</i> <i>500: Serverfehler</i>
<i>api/exhibitions/:id</i>		
GET	Gibt eine spezifische Ausstellung basierend auf der <i>id</i> zurück.	<i>200: Daten der Ausstellung</i> <i>500: Serverfehler</i>
PUT	Aktualisiert die Daten einer bestehenden Ausstellung basierend auf der <i>id</i> .	<i>200: Ausstellung erfolgreich aktualisiert</i> <i>404: Ausstellung nicht gefunden</i> <i>500: Serverfehler</i>
DELETE	Löscht eine spezifische Ausstellung basierend auf der <i>id</i> .	<i>204: Ausstellung erfolgreich gelöscht</i> <i>404: Ausstellung nicht gefunden</i> <i>500: Serverfehler</i>
<i>api/exhibitions/:id/document</i>		
POST	Fügt ein Dokument zu einer Ausstellung basierend auf der <i>id</i> hinzu.	<i>200: Dokument erfolgreich hinzugefügt</i> <i>404: Ausstellung nicht gefunden</i> <i>500: Serverfehler</i>
DELETE	Löscht ein Dokument aus einer Ausstellung basierend auf der <i>id</i> .	<i>200: Dokument erfolgreich gelöscht</i> <i>404: Ausstellung nicht gefunden</i> <i>500: Serverfehler</i>
<i>api/exhibitions/:id/delivery</i>		

POST	Fügt eine Lieferung zu einer Ausstellung basierend auf der <i>id</i> hinzu.	<i>200: Lieferung erfolgreich hinzugefügt</i> <i>404: Ausstellung nicht gefunden</i> <i>500: Serverfehler</i>
PUT	Aktualisiert eine bestehende Lieferung zu einer Ausstellung basierend auf der <i>id</i> .	<i>200: Lieferung erfolgreich aktualisiert</i> <i>404: Ausstellung nicht gefunden</i> <i>500: Serverfehler</i>
DELETE	Löscht eine Lieferung aus einer Ausstellung basierend auf der <i>id</i> .	<i>200: Lieferung erfolgreich gelöscht</i> <i>404: Ausstellung nicht gefunden</i> <i>500: Serverfehler</i>
<i>api/assets</i>		
GET	Gibt eine Liste aller Assets zurück.	<i>200: Liste der Assets</i> <i>500: Serverfehler</i>
POST	Fügt ein neues Asset hinzu.	<i>200: Asset erfolgreich erstellt</i> <i>500: Serverfehler</i>
<i>api/assets/:id</i>		
GET	Gibt ein spezifisches Asset basierend auf der <i>id</i> zurück.	<i>200: Daten des Assets</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
PUT	Aktualisiert ein bestehendes Asset basierend auf der <i>id</i> .	<i>200: Asset erfolgreich aktualisiert</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
DELETE	Löscht ein spezifisches Asset basierend auf der <i>id</i> .	<i>204: Asset erfolgreich gelöscht</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
<i>api/assets/banner</i>		

GET	Gibt eine Liste von Bannern zurück.	<i>200: Liste der Banner</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
<i>api/assets/banner/:id</i>		
GET	Gibt ein spezifisches Banner basierend auf der <i>id</i> zurück.	<i>200: Daten des Banners</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
<i>api/assets/banner-design-element</i>		
GET	Gibt eine Liste von Bannern und Design-Elementen zurück.	<i>200: Liste der Banner und Design-Elemente</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
<i>api/assets/design-asset</i>		
GET	Gibt eine Liste von Design-Objekten und Informationstafeln zurück.	<i>200: Liste der Design-Objekte und Informationstafeln</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
<i>api/assets/design-object/:id</i>		
GET	Gibt ein spezifisches Design-Objekt basierend auf der <i>id</i> zurück.	<i>200: Daten des Design-Objekts</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
<i>api/assets/design-element</i>		
GET	Gibt eine Liste von Design-Elementen zurück.	<i>200: Liste der Design-Elemente</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
<i>api/assets/design-element/:id</i>		
GET	Gibt ein spezifisches Design-Element basierend auf der <i>id</i> zurück.	<i>200: Daten des Design-Elements</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>

<b><i>api/assets/digital-models</i></b>		
GET	Gibt eine gruppierte Liste von digitalen Modellen zurück.	<i>200: Liste der digitalen Modelle</i> <i>500: Serverfehler</i>
<b><i>api/assets/digital-model/:id</i></b>		
GET	Gibt ein spezifisches digitales Modell basierend auf der <i>id</i> zurück.	<i>200: Daten des digitalen Modells</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
<b><i>api/assets/drawing</i></b>		
GET	Gibt eine Liste von Zeichnungen zurück.	<i>200: Liste der Zeichnungen</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
<b><i>api/assets/drawing/</i></b>		
GET	Gibt eine spezifische Zeichnung basierend auf der <i>id</i> zurück.	<i>200: Daten der Zeichnung</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
<b><i>api/assets/mechanical-model</i></b>		
GET	Gibt eine gruppierte Liste von mechanischen Modellen zurück.	<i>200: Liste der mechanischen Modelle</i> <i>500: Serverfehler</i>
<b><i>api/assets/mechanical-model/:id</i></b>		
GET	Gibt ein spezifisches mechanisches Modell basierend auf der <i>id</i> zurück.	<i>200: Daten des mechanischen Modells</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
<b><i>api/assets/mechanical-model/:id/document</i></b>		
POST	Fügt ein Dokument zu einem mechanischen Modell basierend auf der <i>id</i> hinzu.	<i>200: Dokument erfolgreich hinzugefügt</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>

GET	Gibt ein Dokument eines mechanischen Modells basierend auf der <i>id</i> zurück.	<i>200: Dokument erfolgreich zurückgegeben</i> <i>500: Serverfehler</i>
DELETE	Löscht ein Dokument eines mechanischen Modells basierend auf der <i>id</i> .	<i>200: Dokument erfolgreich gelöscht</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
<i>api/assets/mechanical-model/:id/pedestal</i>		
POST	Fügt einen Sockel zu einem mechanischen Modell basierend auf der <i>id</i> hinzu.	<i>200: Sockel erfolgreich hinzugefügt</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
PUT	Aktualisiert einen Sockel eines mechanischen Modells basierend auf der <i>id</i> .	<i>200: Sockel erfolgreich aktualisiert</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
DELETE	Löscht einen Sockel eines mechanischen Modells basierend auf der <i>id</i> .	<i>200: Sockel erfolgreich gelöscht</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
<i>api/assets/mechanical-model/:id/accessory</i>		
POST	Fügt ein Zubehör zu einem mechanischen Modell basierend auf der <i>id</i> hinzu.	<i>200: Zubehör erfolgreich hinzugefügt</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
PUT	Aktualisiert ein Zubehör eines mechanischen Modells basierend auf der <i>id</i> .	<i>200: Zubehör erfolgreich aktualisiert</i> <i>404: Objekt nicht gefunden</i>
DELETE	Löscht ein Zubehör aus einem mechanischen Modell basierend auf der <i>id</i> .	<i>200: Zubehör gelöscht</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
<i>api/assets/mechanical-model/:id/packing</i>		

POST	Fügt eine Verpackung zu einem mechanischen Modell basierend auf der <i>id</i> hinzu.	<i>200: Verpackung erfolgreich hinzugefügt</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
DELETE	Löscht eine Verpackung aus einem mechanischen Modell basierend auf der <i>id</i> .	<i>200: Sockel gelöscht</i> <i>404: Objekt nicht gefunden</i> <i>500: Serverfehler</i>
<i>/api/codices</i>		
GET	Ruft alle Codices ab	<i>200: Erfolgreiche Abfrage</i> <i>500: Serverfehler</i>
POST	Fügt einen neuen Codex hinzu	<i>200: Codex erfolgreich erstellt</i> <i>400: Codex existiert bereits</i> <i>500: Serverfehler</i>
<i>/api/codices/:id</i>		
GET	Ruft einen Codex basierend auf der <i>id</i> ab	<i>200: Codex erfolgreich abgerufen</i> <i>500: Serverfehler</i>
PUT	Aktualisiert einen Codex basierend auf der <i>id</i>	<i>200: Codex erfolgreich aktualisiert</i> <i>404: Codex nicht gefunden</i> <i>500: Serverfehler</i>
DELETE	Löscht einen Codex basierend auf der <i>id</i>	<i>204: Codex erfolgreich gelöscht</i> <i>404: Codex nicht gefunden</i> <i>500: Serverfehler</i>
<i>api/codices/:id/pages</i>		
GET	Ruft alle Seiten eines bestimmten Codex basierend auf der <i>id</i> ab	<i>200: Erfolgreiche Abfrage</i> <i>404: Codex nicht gefunden</i> <i>500: Serverfehler</i>
POST	Fügt eine neue Seite zu einem bestimmten Codex hinzu	<i>200: Seite erfolgreich hinzugefügt</i> <i>400: Seite existiert bereits</i>

		<i>404: Codex nicht gefunden</i> <i>500: Serverfehler</i>
<i>api/codices/:id/page/:pageId</i>		
GET	Ruft eine bestimmte Seite eines Codex basierend auf der <i>id</i> und <i>pageId</i> ab	<i>200: Erfolgreiche Abfrage</i> <i>404: Codex oder Seite nicht gefunden</i> <i>500: Serverfehler</i>
PUT	Aktualisiert eine Seite eines Codex basierend auf der <i>id</i> und <i>pageId</i>	<i>200: Seite erfolgreich aktualisiert</i> <i>404: Codex oder Seite nicht gefunden</i> <i>500: Serverfehler</i>
DELETE	Löscht eine Seite eines Codex basierend auf der <i>id</i> und <i>pageId</i>	<i>204: Seite erfolgreich gelöscht</i> <i>404: Codex oder Seite nicht gefunden</i> <i>500: Serverfehler</i>

Tab. 10: Endpunkte der REST-API

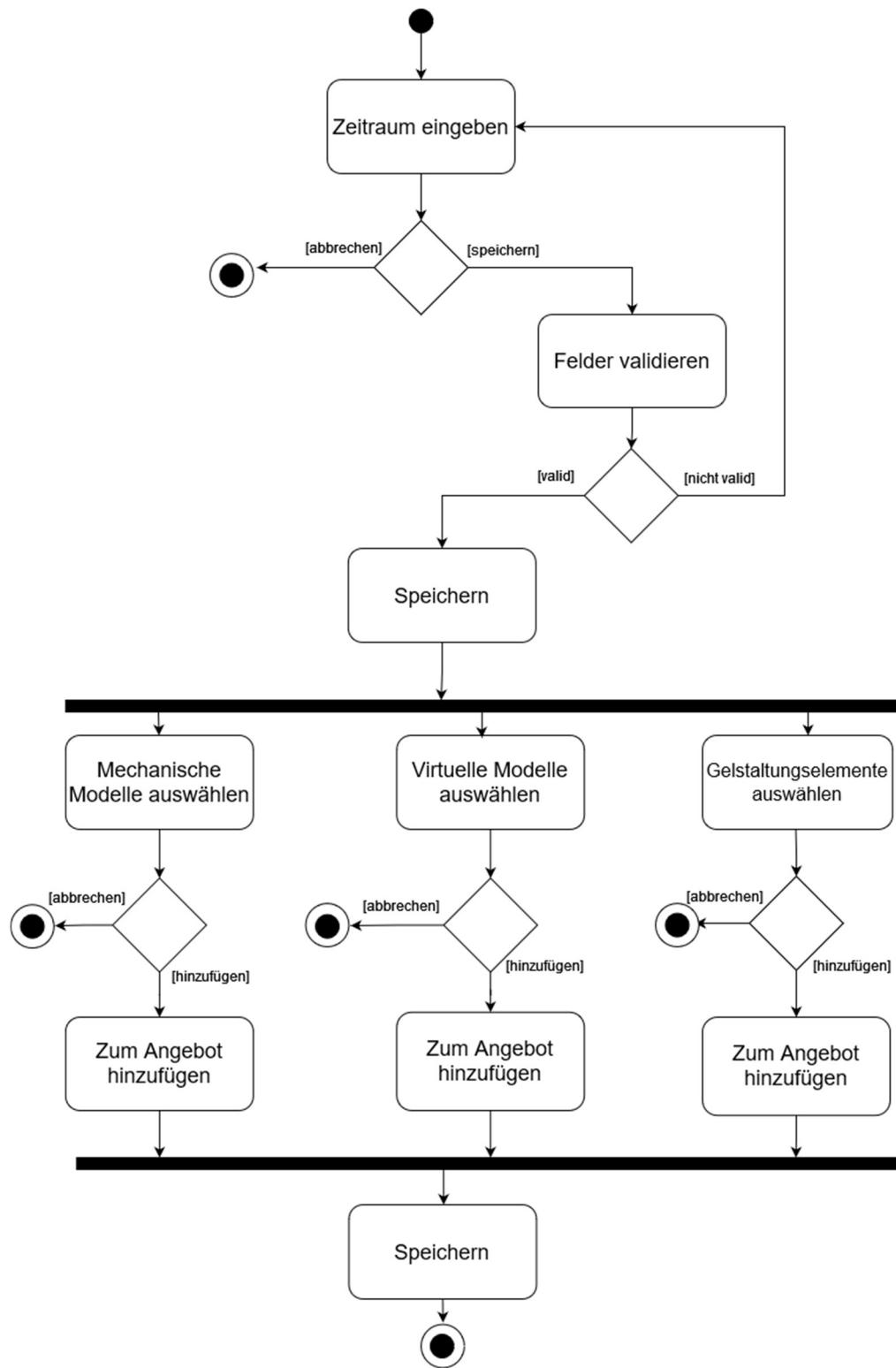


Abb. 32: Aktivitätsdiagramm zum Erstellen eines Angebots für einen Kunden.

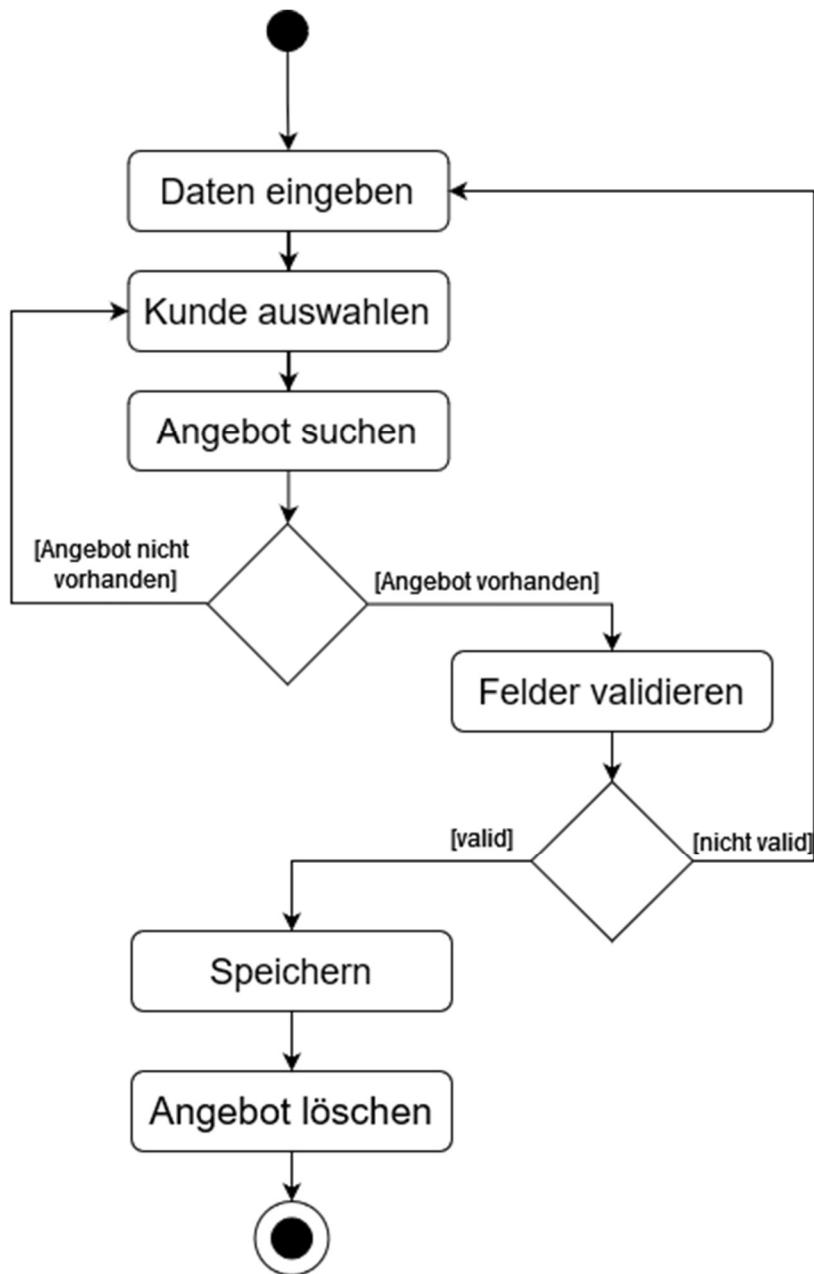


Abb. 33: Aktivitätsdiagramm zum Erstellen einer Ausstellung