

Hochschule Osnabrück

University of Applied Sciences

Fakultät

Ingenieurwissenschaften und Informatik

Schriftliche Ausarbeitung zum Thema:

**Realisierung eines virtuellen Kleiderschranks mittels einer API
mit Quarkus**

im Rahmen des Moduls

Software-Architektur – Konzepte und Anwendungen,
des Studiengangs Informatik-Technische Informatik

Autor:	Manuel Arling
Matr.-Nr.:	965366
E-Mail:	manuel.arling@hs-osn- mabrueck.de
Autor:	Thomas Meese
Matr.-Nr.:	965321
E-Mail:	thomas.meese@hs-osnab- rueck.de
Themensteller:	Prof. Dr. Rainer Roosmann

Abgabedatum: 17.02.2023

Inhaltsverzeichnis

Inhaltsverzeichnis.....	2
Abbildungsverzeichnis	4
Tabellenverzeichnis.....	Fehler! Textmarke nicht definiert.
Source-Code Verzeichnis.....	Fehler! Textmarke nicht definiert.
Abkürzungsverzeichnis	5
1 Einleitung (Manuel Arling)	1
1.1 Vorstellung des Themas	1
1.2 Ziel der Ausarbeitung	1
1.3 Aufbau der Hausarbeit.....	1
2 Darstellung der Grundlagen	2
2.1 Quarkus (Manuel Arling)	2
2.2 Quarkus Qute (Manuel Arling)	2
2.3 Bootstrap (Manuel Arling)	2
2.4 JPA und JTA (Thomas Meese)	2
2.5 CDI (Thomas Meese)	4
2.6 OAuth2.0 OIDC (Thomas Meese)	4
2.7 OIDC (Thomas Meese)	5
2.8 Keycloak (Thomas Meese).....	6
2.9 Schichtenarchitektur und ECB-Pattern (Manuel Arling)	7
2.10Domain-driven Design (Manuel Arling).....	7
2.11Richardson Maturity Modell (Manuel Arling).....	8
2.12HATEOAS (Manuel Arling)	9
3 Anwendung.....	10
3.1 Aufbau des Programms	10
3.2 Boundary (Manuel Arling).....	10
3.3 Swagger-UI/OpenAPI (Manuel Arling).....	12
3.3.1 Daten Transfer Objekte (Manuel Arling)	12
3.3.2 Rest (Manuel Arling)	13
3.3.3 Web (Manuel Arling).....	14
3.4 Control (Manuel Arling)	15
3.5 Anti-Corruption Layer (Manuel Arling)	16
3.5.1 Interne ACL (Thomas Meese)	16
3.5.2 Externe ACL (Manuel Arling).....	17
3.6 Authentifizierung der Nutzer (Thomas Meese).....	19
3.6.1 Umsetzung von Keycloak	19
3.6.2 Workaround mit dem Keycloak Admin-Client.....	20
3.7 Entitys (Thomas Meese).....	22
3.8 Gateways (Thomas Meese)	22
3.8.1 Events	23
3.9 Ausfallsicherheit (Thomas Meese).....	23
3.10Frontend.....	24
3.10.1 Generelle Ansicht (Thomas Meese).....	24

3.10.2	Forms und Detail Ansicht (Manuel)	25
3.10.3	Direktlinkseiten (Thomas Meese)	28
3.11	Qualität	29
3.11.1	Developer Experience und User Experience.....	30
3.11.2	Korrektheit	30
3.11.3	Sicherheit.....	30
3.11.4	Resilienz.....	30
4	Zusammenfassung und Fazit	31
4.1	Fazit 31	
5	Referenzen.....	32

Abbildungsverzeichnis

Abbildung 1 Auhorization Code Flow ([@OAu] S.24)5

Abbildung 2 ECB-Pattern [vgl. @BCE] 7

Abbildung 3 Richardson Maturity Model all Levels8

Abbildung 4 DTO: KleidungsstueckInputDTO 15

Abbildung interne ACL 16

Abbildung 5 Hauptansicht Meine Kleidungsstücke24

Abbildung 6 Modal zum Erstellen von Kleidungsstücken.....25

Abbildung 7 Detailansicht Outfit26

Abbildung Detailansicht Outfit26

Abbildung Bearbeitungsansicht Outfit27

Abbildung Konto anlegen29

Abkürzungsverzeichnis

CDI	Context and Dependency Injection for the Java EE Plattform
DTO	Daten Transfer Objekt
ECB	Entity-Controller-Boundary Pattern
Java EE	Java Enterprise Edition, in der Version 7
JSF	Java Server Faces
SWA	Software-Architektur
API	Application Programming Interface
UI	User Interface
BFF	Backend-for-Frontend Pattern
REST	Representational State Transfer
HATEOAS	Hypermedia as the Engine of Application State
SSO	Single Sign On
JPA	Java Persistence API
JTA	Java Transaction API
OAuth2	Open Authorization 2.0
OIDC	Open Id Connect
JWT	Jason Web Token
SQL	Simple Query Language
ORM	Object Relational Mapping
JDBC	Java Database Connectivity
SAML	Security Assertion Markup Language
IAM	Identity and Access Management
URI	Uniform Ressource Identifier
DDD	Domain-driven Design

1 Einleitung (Manuel Arling)

Die Softwarearchitektur befasst sich mit der Organisation und Strukturierung von Programmen. Dabei werden relevante Design-Aspekte definiert und festgehalten. Das Thema der Softwarearchitektur stellt ein sehr großes und breitgefächertes Gebiet dar. Damit ein Programm eine geeignete Softwarearchitektur besitzt, ist es von Nöten viele Konzepte zu kennen. Die Hausarbeit beschäftigt sich dabei mit genau diesem Aspekt.

1.1 Vorstellung des Themas

Diese Hausarbeit befasst sich nämlich mit der Entwicklung einer RESTful API, welche die einzelnen Aspekte der Softwarearchitekturen aufgreift und umsetzt. Dabei sind bedeutsame Patterns und Konzepte aufzunehmen, die die Komplexität der API reduziert. Es ist ebenfalls zu sehen, dass die Entwicklung von komplexen Programmen immer stärker in Richtung der Entwicklung eines Programms mit kleineren voneinander unabhängigen Modulen. [vgl. @MIC] Auch dieser Aspekt soll aufgegriffen werden und auf die Anwendbarkeit für diese Anwendung geprüft werden. Zusätzlich besitzt die Qualität einer API große Bedeutung. Damit diese erreicht werden kann, müssen Standards festgelegt werden, welche in der Ausarbeitung eingehalten werden sollen. Einer dieser stellt der REST-Architekturstil dar.

1.2 Ziel der Ausarbeitung

Mit dem Projekt wird konkret das Ziel verfolgt eine API unter Zuhilfenahme des Quarkus-Frameworks zu entwickeln und zu implementieren. Bei der API handelt es sich um einen virtuellen Kleiderschrank, der es Nutzern ermöglichen soll, ihre Kleidung digital zu verwalten. Dabei haben Nutzer die Möglichkeit Kleidungsstücke dem Kleiderschrank hinzuzufügen, zu bearbeiten oder zu entfernen. Ebenfalls soll den Nutzern ermöglicht werden mit den Kleidungsstücken Outfits zu erstellen und diese zu verwalten (bearbeiten, löschen). Neben der Möglichkeit selbst Kleidungsstücke dem Kleiderschrank hinzuzufügen, soll mindestens eine externe API via Rest-Client eingebunden werden, womit der Anwender Kleidungsstücke, die er Online gekauft hat (z.B. via H&M etc.), automatisch per Artikelnummer dem Kleiderschrank hinzufügen kann. Dazu kann ein Nutzer seine erstellten Outfits teilen und somit anderen Nutzern lesenden Zugriff auf sein Outfit erlauben umso z.B. auch für Kleidungsstücke eines Händlers zu werben. Ziel dabei ist es den Nutzer die zwei Schnittstellen Rest-Anwendung und Web-Anwendung zur Verfügung zu stellen.

1.3 Aufbau der Hausarbeit

Die weiteren Kapitel der Hausarbeit befassen sich mit der Darstellung der wichtigsten Frameworks, Konzepte und Stile. Dies wird benötigt, da die in dem darauffolgenden Kapitel dargestellte Anwendung die genannten Frameworks, Konzepte und Stile aufgreift und umsetzt. Die Anwendung lässt sich dabei in kleinere Teile unterteilen, welches eine bessere Struktur der Anwendung ermöglicht.

2 Darstellung der Grundlagen

Dieses Kapitel gibt einen Überblick über die wichtigsten Themen, welche in der zu entwickelnden API aufgegriffen werden.

2.1 Quarkus (Manuel Arling)

Quarkus ist ein Open-Source-Framework, welches für Java Anwendungen entwickelt wurde. Es ist für Cloud- und Container-Umgebungen spezialisiert, da es von Grund auf für Kubernetes realisiert wurde. Es bietet Unterstützungen für die Entwicklung von Microservices, Rest und Reactive Programmierung. Auch Unterstützungen für Datenbanken sind vorhanden. Dadurch lassen sich Programme erstellen, welche in kleinen Containern verwendet werden können. [vgl. @QUA]

2.2 Quarkus Qute (Manuel Arling)

Quarkus Qute stellt eine Template Engine dar. Eine Template Engine ermöglicht es dabei durch die Erstellung von Vorlagen (im weiteren Verlauf als Template bezeichnet) die Möglichkeit durch die Verwendung von Platzhaltern in der Vorlage generisch Seiten zu erstellen. Dabei werden die Platzhalter mit aktuellen Inhalten gefüllt. Zusätzlich können oft auch Anweisungen und Schleifen verwendet werden. [vgl. @TEM]

Quarkus Qute gehört zum Quarkus-Framework und ist somit speziell für Quarkus ausgelegt. Qute setzt die genannten Funktionalitäten einer Template Engine um. Zusätzlich ermöglicht Qute die Verwendung von typsicheren Templates, dadurch können Typüberprüfungen zur build-Zeit stattfinden [vgl. @OUT]. Es ermöglicht ebenfalls das Einbetten eines Templates in ein anderes Template. Dadurch könnten die Webseiten in kleine Komponenten aufgetrennt werden und stellen somit wiederverwendbare Elemente dar, was die Wartbarkeit verbessert.

2.3 Bootstrap (Manuel Arling)

Bootstrap ist ebenfalls ein Open-Source-Framework, welches für Webentwickler entwickelt wurde. Es ermöglicht mit minimalem Aufwand responsive und ansprechende Webseiten zu gestalten. Dafür werden von Bootstrap zur Verfügung gestellte Komponenten verwendet, wie z.B. die Navbar, Modals und Buttons. Es verwendet dabei HTML, CSS und Javascript. [vgl. @BOO]

2.4 JPA und JTA (Thomas Meese)

JPA wird die simplere Interaktion mit Datenbanken verwendet. Innerhalb der API sind Komponenten, Annotationen und Verhaltensweisen definiert, die eine ORM-Schicht realisieren. Dabei ist das Ziel von ORM-Datenbank-Schichten so weit zu Abstrahieren, das ein Verwenden von herkömmlichen SQL in Verbindung mit JDBC Java-Klassen obsolet wird. Sowohl für den lesenden als auch den schreibenden Zugriff wird der EntityManager benötigt, der die dafür vorgesehenen Funktionen beinhaltet (vgl.[TK20] S.190 f.). Der

Kommentiert [TM1]: Quelle: Beginning Quarkus Framework
Seite 190-194

lesende Zugriff auf Datenbanken über TypedQuery deren Ergebnisse über das Result-Set abgerufen werden kann. Das erstellen und bearbeiten von Objekten erfolgt über die *persist()* Methode. Diese ermöglicht es Objekt erstmals in der Datenbank abzulegen und nach einem Bearbeiten des Objektes dieses zu aktualisieren. Über die *delete()*-Methode können Objekte wieder aus der Datenbank entfernt werden.

```
@Entity
@Table(name = "Studis")
public class Student {
    @Id
    @GeneratedValue
    private long matrikelnummer;

    @Column(name = "Firstname")
    private String vorname;
    private String nachname;

    ...sonstige Attribute und Methoden der Klasse
}
```

Snippet 1 Aufbau der Beispielklasse Student

Im Snippet ist eine beispielhafte Annotation einer Java-Klasse erkennbar. Die zuvor erwähnten Objekte werden Entitäts genannt und können über `@Entity` Annotation gekennzeichnet werden. Über die `@Table` Annotation kann innerhalb der Datenbank ein Name für die Tabelle festgelegt werden. Da die Entitäts in der Datenbank eindeutig identifizierbar sein sollen, müssen sie über einen Primary Key verfügen. Dieser kann durch die Annotation `@Id` gesetzt werden (im Beispiel wäre die Matrikelnummer des Studenten der Primary Key). Mittels der `@GeneratedValue` Annotation kann festgelegt werden, dass die Ids automatisch generiert werden sollen. Ebenfalls ist es möglich einzelne Spalten der Tabelle zu benennen, was mittels `@Column` geschieht. Die aus SQL-bekannten Relationen zwischen Entitäts können hier über die Annotationen `@OneToOne`, `@OneToMany` und `@ManyToMany` abgebildet werden. Zusätzlich gibt es die Möglichkeit Relationen zwischen Entitäts und Listen von Objekten wie `list<String>` über ein `@EnumeratedValue` einzubinden. Bei den Relationen kommt auch der sog. *fetchType* zum Einsatz. Dieser legt fest, ob ein Feld mit der zugehörigen Entitätsinstanz sofort gemapped werden soll oder erst wenn das jeweilige Feld verwendet wird. Für die gewünschten ACID-Eigenschaften werden die JTA-Annotationen verwendet, dabei kennzeichnet `@Transactional` eine Transaktion. Über das Value Attribut kann die Transaktion spezifiziert werden. Ein „Required“ sagt beispielsweise aus, dass eine Transaktion gestartet werden muss, sobald die Methode betreten wird. zusätzlich können über das Rollback Attribut Fehlerfälle definiert werden, bei denen ein Rollback auf der Datenbank erfolgen soll (z.B. bei einer `SQLException`). Dennoch gibt es auch einige Dinge bei den Transaktionen zu beachten. So können sie beispielsweise bei statischen Methoden und Lebenszyklus Methoden (`@Postconstruct`) nicht verwendet werden. (vgl. [TKJS.191ff])

2.5 CDI (Thomas Meese)

Kommentiert [TM2]: <https://quarkus.io/guides/cdi>

CDI ist eines der Standard Dependency Injection Frameworks, welches mit Java EE 6 eingeführt wird. Es ermöglicht Entwicklern eine einfache Steuerung von Lebenszyklen ihrer Objekte. Dabei wird der Lebenszyklus nicht mehr vom Entwickler selbst, sondern wird von einem Container übernommen. Abhängigkeiten zu anderen Objekten bzw. Klassen werden zur Laufzeit festgelegt. Benötigt beispielsweise eine Klasse eine andere für die Initialisierung, erfolgt dies über eine Injizierung via Annotation. (vgl. `@CDI`)

2.6 OAuth2.0 OIDC (Thomas Meese)

Kommentiert [TM3]: Quelle: <https://www.rfc-editor.org/rfc/rfc6749.html>

OAuth ist ein Autorisierungsprotokoll, welches seinen Vorgänger im Jahr 2012 abgelöst hat. Es ermöglicht Drittanbietern eingeschränkten Zugang zu Ressourcen von Diensten wie APIs zu erhalten. Es wird dabei festgelegt welche Aktionen der Client ausführen darf, ohne dass eine Weitergabe der Authentifizierungsdaten erfolgt. Die Struktur von OAuth gliedert sich dabei in vier Rollen.

Kommentiert [TM4]: Ins Abkürzungsverzeichnis aufnehmen

Kommentiert [TM5]: Besseres Wort

- Ressourcenbesitzer: Besitzer der geschützten Ressource der Zugriff auf diese gewähren kann
- Ressourcen-Server: Server der die Ressource bereitstellt. Dieser ist dazu in der Lage Anfragen auf die geschützte Ressource, unter Zuhilfenahme von Zugriffstokens, zu akzeptieren und zu beantworten
- Client: Eine Anwendung die eine Anfrage auf die geschützte Ressource im Auftrag des Ressourcenbesitzers im Rahmen seiner Autorisierung tätigt.
- Autorisierungsserver: Der Server stellt dem Client den Zugriffstoken nach erfolgreicher Authentifizierung des Ressourcenbesitzers und anfordern der Autorisierung (vgl. `@OAuth` S.5)

Der Zugriff auf eine geschützte Ressource hat dabei einen festen generellen Ablauf. Die Anwendung fordert vom Ressourcenbesitzer die Autorisierung für den Zugriff an. Der Ressourcenbesitzer gewährt die Autorisierung in Form eines Berechtigungsnachweises (credential) über einen der vier möglichen Wege. Die Anwendung fordert nun vom Autorisierungsserver ein Zugriffstoken an, indem er sich bei diesem authentifiziert und die Zugriffsberechtigung präsentiert. Nach der erfolgreichen Authentifizierung der Anwendung und Validierung der Berechtigung erhält die Anwendung den Token. Die Anwendung fragt nun beim Ressourcen-Server die geschützte Ressource an, indem er sich durch den Token authentifiziert. Bei erfolgreicher Validierung des Tokens antwortet der Server. (vgl. `@OAuth` S.7. f.)

Wie bereits erwähnt gibt es vier mögliche (nicht veraltete) Varianten einer Anwendung die Autorisierung zu erteilen. Diese sind der Authorization Code, der Implicit Grant, die Passwort Credentials und die Client Credentials. Da im späteren System die Variante Authorisation Code verwendet werden soll, wird diese nun näher betrachtet.

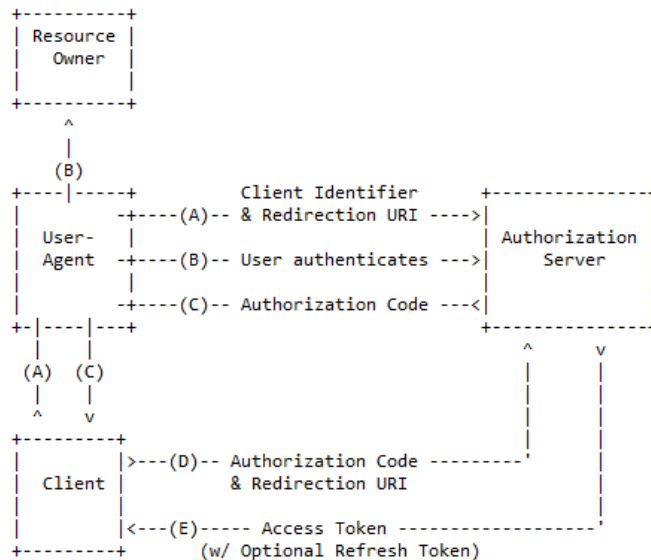


Abbildung 1 Authorization Code Flow ([@OAU] S.24)

In Abbildung 1 ist der Ablauf des Authorizations Codes visualisiert. Der Authorization Code wird benutzt, um Access Token und Refresh Token zu erhalten. Die Anwendung startet den Prozess in dem er den User Agent (in der Regel ein Web-Browser) des Ressourcenbesitzers zum Autorisierungsserver. Er gibt dabei seine ID, seinen Status, den Geltungsbereich und die redirect URI zu der der Agent anschließend zurückgeschickt wird. Der Server validiert die Anfrage und gewährt bzw. verwehrt die Zugriffsanfrage der Anwendung. Wenn der Zugriff gewährt wurde, wird der Agent zurück zur Anwendung geleitet, die URI enthält dabei den Authorization Code. Nun kann die Anwendung einen Accesstoken vom Autorisierungsserver anfordern, indem er den Authorization Code verwendet. Bei einer erfolgreichen Validierung antwortet der Server mit dem Zugriffstoken. (vgl. [@OAU]. S24 f.)

2.7 OIDC (Thomas Meese)

OIDC ist ein Protokoll Identitätsauthentifizierungsprotokoll welches die Prinzipien von OAuth2.0 verwendet. Es kann für ein Reihe von Anwendungen, wie Apps verwendet werden und unterstützt auch SSO. Dabei wird der Ansatz verfolgt die Anmeldedaten der Nutzer nicht weiterzugeben. Eine der Haupteerweiterungen von OAuth ist dabei die Bereitstellung von ID-Token. Dieses wird über JWT bereitgestellt und enthält Identitätsdaten zu der durchgeführten Authentifizierung enthält. Für die Authentifizierung können drei Varianten verwendet werden. Bei den Authentication Code Flow erhält die Anwendung einen Authentication Code, den dieser dann gegen das ID-Token und den Zugriffstoken bei Autorisierungsserver tauschen kann. Dadurch wird kein Token an den Agent und andere mögliche Anwendungen die Zugriff auf diesen haben könnten weiter-

Kommentiert [TM6]: Quelle: https://openid.net/specs/openid-connect-core-1_0-final.html

Kommentiert [TM7]: Ins Abkürzungsverzeichnis aufnehmen

geben. Beim Impliziten Flow wird der Agent von Autorisierungsserver direkt mit den Tokens zur Anwendung (nach erfolgreicher Validierung) zurückgesendet. Der Zwischenschritt, welcher den Code beinhaltet fällt hier also weg. In der letzten Variante werden einige Tokens vom Autorisierungsserver direkt und andere vom Token Endpoint erhalten. (vgl. [@OID])

2.8 Keycloak (Thomas Meese)

Keycloak ist ein open source IAM-Tool mit einem Fokus auf moderne Anwendungen wie Rest-APIs. Keycloak unterstützt sowohl SSO als auch Session Management. So haben beispielsweise Nutzer und Admins Einsicht bei welcher Anwendung der Nutzer authentifiziert ist und haben die Möglichkeit die Sitzung manuell zu beenden. Für die Authentifizierung verwendet Keycloak die Protokolle OIDC, OAuth und SAML. Keycloak kann über verschiedene Arten verwendet werden. Die hier verwendete Variante bindet Keycloak über Maven in das Quarkus-Projekt ein, wodurch es über die Konfig-Datei eingerichtet werden kann. (vgl. [STPS21] S. 4-5)

```
%dev.quarkus.keycloak.devservices.enabled=true
%prod.quarkus.keycloak.devservices.enabled=false
quarkus.keycloak.devservices.port=8180
quarkus.keycloak.devservices.grant.type=code
quarkus.oidc.application-type=web-app
quarkus.keycloak.devservices.users.maxmustermann=kreativ
quarkus.keycloak.devservices.roles.maxmustermann=user
```

Snippet 2 Ausschnitt aus der Keycloak Konfig

Snippet 2 zeigt eine Minimale Konfiguration innerhalb der application.properties der Quarkus Anwendung um Keycloak verwenden zu können. Das für die weitere Konfiguration das Admin-Terminal verwendet werden kann wir für den Server ein fester Port festgelegt, somit ist es unter der URL <http://localhost:8180> erreichbar. Über den grant type kann der Authorization Grant Type von OAuth festgelegt werden. Während über den Application-type die entsprechende Einstellung für OIDC vorgenommen werden kann. Ebenfalls kann ein oder mehrere Default Nutzer angelegt werden. In diesem Fall existiert ein Nutzer, der die Rolle „user“ besitzt und sich mit dem Benutzernamen: „maxmustermann“ und dem Passwort: „kreativ“ anmelden kann. Für die Nutzer stellt Keycloak bereits eine eigene Datenbank zur Verfügung, daher ist es nicht zwingend nötig. Jedoch besteht die Möglichkeit bereits vorhandene Infrastruktur für die Nutzerdatenbank zu verwenden.

Kommentiert [TMS]: Quelle:
https://books.google.de/books?hl=de&lr=&id=WBs-vEAAQBAJ&oi=fnd&pg=PP1&dq=keycloak&ots=npJ4VH1BS-&sig=uTJNPg-CiYT3u14_BJgJer82g24&redir_esc=y#v=one-page&q=keycloak&f=false

2.9 Schichtenarchitektur und ECB-Pattern (Manuel Arling)

Bei der Schichtenarchitektur wird ein Programm in logische horizontale Schichten unterteilt. Jede dieser Schichten besitzt eine bestimmte Rolle. Es ermöglicht große Anwendungen zu strukturieren und klare Grenzen zu definieren.

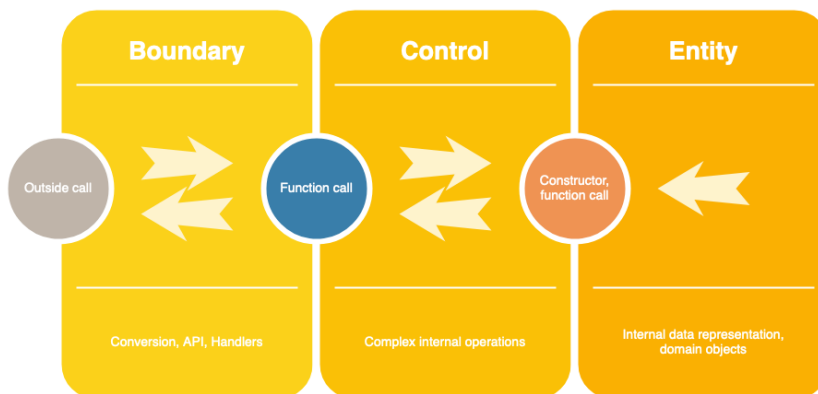


Abbildung 2 ECB-Pattern [vgl. @BCE]

Das ECB-Pattern stellt ein Pattern dar, welches die Schichtenarchitektur umsetzt. In Abbildung 2 ist zu erkennen, dass das Pattern ein Programm in die drei Schichten Entity, Control und Boundary aufteilt. Dabei zielt das ECB-Pattern darauf ab, dass jede Schicht unabhängig voneinander ist. Sprich ändert sich die Boundary Schicht, so muss an der Control Schicht nichts angepasst werden. Dadurch eine unabhängige Programmierung einzelner Schichten ermöglicht.

Dabei übernehmen die einzelnen Schichten folgende Aufgaben:

- Boundary: Die Schicht stellt die Schnittstelle zwischen dem System und dem Benutzer oder einem externen System dar.
- Control: Die Control-Schicht repräsentiert die Applikationslogik dar. Dort befinden sich oft Berechnungen und Algorithmen.
- Entity: Die Entity-Schicht beinhaltet alle Datenstrukturen der Applikation und hält somit die Daten der Anwendung. Diese stellt oftmals eine Datenbank dar.

2.10 Domain-driven Design (Manuel Arling)

Domain-driven Design (auch DDD genannt) ist kein reines Konzept für die Softwareentwicklung dar. Es stellt viel mehr Design-Ansatz mit einer Sammlung von Methoden dar. Dabei steht die Domäne im Mittelpunkt. Eine Domäne stellt dabei ein Anwendungsgebiet dar, oft auch als Fachlichkeit bezeichnet. Dessen Modellierung ist von großer Bedeutung

Kommentiert [MA9]: https://adambien.blog/roller/adambien/entry/web_components_boundary_control_entity

Kommentiert [MA10R9]: <https://vaclavkosar.com/software/Boundary-Control-Entity-Architecture-The-Pattern-to-Structure-Your-Classes>

Kommentiert [MA11R9]: O'Reilly S.139

bei dem Domain-driven Design. Dabei werden komplexe Geschäftsprozesse in klar definierten Modellen und Begriffen abgebildet. Ein paar dieser Modelle stellen Aggregate, Repositories und Service-Klassen dar. [vgl. @DDD]

2.11 Richardson Maturity Model (Manuel Arling)

Kommentiert [MA12]: <https://martinfowler.com/articles/richardsonMaturityModel.html>

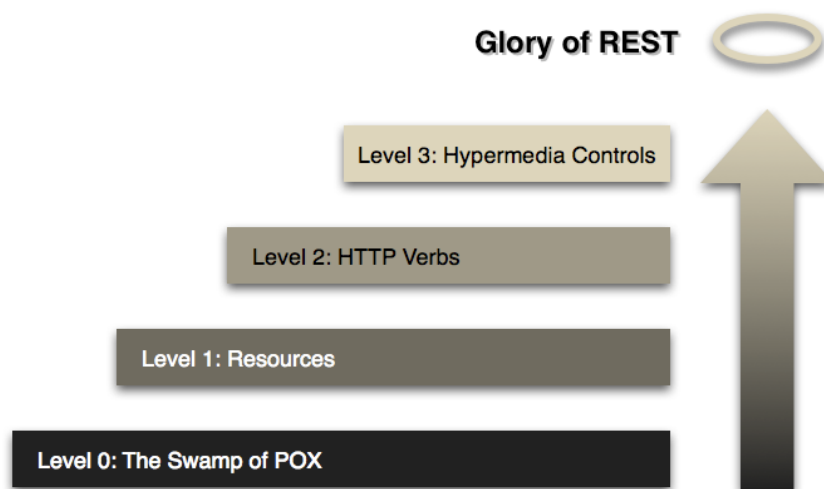


Abbildung 3 Richardson Maturity Model all Levels

Abbildung 3 zeigt das Richardson Maturity Modell auf, welche die grundlegenden Prinzipien des REST-Paradigmas in vier Ebenen aufteilt. Es stellt die Reife einer Webanwendung dar. Die vier Stufen lassen sich wie folgt beschreiben:

- Level 0 "The Swamp of POX": Definiert Webanwendungen, welche die HTTP-Technologie verwendet, jedoch keiner der möglichen Strukturierungsmöglichkeiten umsetzt.
- Level 1 „Ressource“: Bei dieser Stufe werden URI-basierte Ressourcen verwendet. Sprich es stehen mehrere Endpunkte zur Verfügung.
- Level 2 „HTTP Verbs“: Stufe 2 erweitert die vorherige Stufe damit, dass die HTTP-Verben GET, POST, PUT, DELETE, etc. verwendet werden.
- Level 3 „Hypermedia Controls“: Dies Stufe stellt die höchste Stufe dar und erweitert die Webanwendung um die Umsetzung von HATEOAS, welches in dem nächsten Kapitel genauer erklärt wird.

Je höher das Level ist, desto reifer ist eine Webanwendungen. Es ermöglicht damit eine geordnete Struktur Objekte und deren Interaktion. [vgl. @RMM]

2.12 HATEOAS (Manuel Arling)

HATEOAS steht für „Hypermedia as the Engine of Application State“ und stellt ein Konzept dar, welches dem Rest-Client die Möglichkeit bietet allein durch das Folgen von URIs durch die Anwendung zu navigieren. Das bedeutet, dass der Client die API ohne eine zusätzliche Dokumentation in der Lage ist alle möglichen Funktionalitäten der Anwendung zu verwenden. Damit dies gelingt, wird dem Client bei einer Anfrage in der Antwort zu der eigentlichen Nutzlast eine Liste von Links zurückgegeben, welche die weiteren Interaktionen mit der Ressource beinhaltet. So erlangt der Client allein durch die Antwort vom Server, was für Funktionen der Server für diese Ressource zur Verfügung stellt. Da HATEOAS im Richardson Maturity Modell die höchste Stufe darstellt, spielt es einen wichtigen Aspekt in dem RESTful-Architekturstil dar. [vgl. @HAT]

3 Anwendung

Dieses Kapitel ist so zu gestalten, wie es zur Aufgabenstellung passt. Überlegen Sie sich eine sinnvolle Strukturierung.

3.1 Aufbau des Programms

Der virtuelle Kleiderschrank dient den Anwender dazu seine Kleidung digital zu katalogisieren und so einen einfachen Überblick über alle Kleidungsstücke zu erhalten. Ebenfalls hat er die Möglichkeit aus den ihm zur Verfügung stehenden Kleidungsstücken Outfits zusammenzustellen. Die Anwendung selbst gliedert sich mithilfe des Domain-driven Designs in zwei Subdomänen. Dabei handelt es sich um das Kleidungsstück und das Outfit. Auch wenn Outfits Kleidungsstücke zwingend benötigen, herrscht keine direkte Abhängigkeit zwischen den beiden Domänen. Eine Lose Kopplung zwischen den einzelnen Subdomänen steht dabei im Vordergrund. Dies wird unter anderem durch die Abschirmung der Schnittstellen zwischen den Modulen voneinander mittels ACLs erreicht. Intern ist jedes Subsystem nach dem Prinzip der Hexagonalen Architektur in Kombination mit ECB-Entwurfsmuster entwickelt. Jede einzelne Schicht agiert dabei unabhängig von den anderen eine Kommunikation zwischen den Ebenen erfolgt nur über austauschbare Schnittstellen. Dies ermöglicht eine hohe Wartbarkeit Änderungen in einer Schicht keine Auswirkungen auf die Funktionalität des restlichen Systems hat. Zusätzlich kommt hier auch noch das Dependency Inversion Prinzip, welches einen wichtigen Bestandteil für die Entwicklung loser voneinander gekoppelter Anwendungen darstellt.

3.2 Boundary (Manuel Arling)

Die Boundary-Schicht des ECB-Design Patterns stellt die Schnittstelle für externe Systeme dar. Durch diese wird festgelegt, wie mit dem System zu interagieren ist. Damit externe Systeme über die Spezifikationen der Kommunikation Informationen erhalten, ist es von großer Bedeutung diese zu dokumentieren und zur Verfügung zu stellen. Dies beeinflusst positiv die Developer Experience. Um eine positive Developer Experience zu gewährleisten, wird für diese Api die Erweiterung OpenApi verwendet.

Es sind die zwei Unterbereiche Rest und Web für die Boundary-Schicht definiert. Dies liegt dem Backend-for-Frontend (BFF) Entwurfsmuster zugrunde. Denn bei dem BFF Pattern wird für jede Frontend-Anwendung eine spezifische Backend-Schnittstelle zur Verfügung gestellt. Dies hat den Vorteil, dass es eine bessere Kommunikation zwischen Frontend und Backend ermöglicht, denn so kann das Backend spezielle Anforderungen und Änderungen einer Frontend-Anwendung keinen Einfluss auf eine andere Frontend-Anwendung. Dadurch werden die Flexibilität und Skalierbarkeit gesteigert.

```
@GET
@Transactional(value = javax.transaction.Transactional.TxType.REQUIRES_NEW)
@RolesAllowed("benutzer")
@Operation(summary = "Holt alle Kleidungsstuecke des eingeloggten Benutzers.", description = "Holt alle vom Benutzer erstellten Kleidungsstuecke aus seinem virtuellen Kleiderschrank mit den gesetzten Filtern.")
@APIResponses(
    value = {
        @APIResponse(
            responseCode = "200",description = "OK",
            content = @Content(mediaType = MediaType.APPLICATION_JSON,
                schema = @Schema(type= SchemaType.ARRAY, implementation = KleidungsstueckListeOutputDTO.class)),
        @APIResponse(
            responseCode = "500",
            description = "Internal Server error"
        )
    }
)
public Response getAlleKleidungsstuecke(KleidungsstueckFilter filter) {...}
```

Snippet 3 Exemplarische Annotationen an einer Ressource-Methode

Das obenstehende Snippet zeigt dabei ein Beispiel für den strukturellen Aufbau einer Methode innerhalb der Ressource Klasse auf. Die dargestellten Annotationen finden in nahezu jeder Ressourcen-Methode Anwendung. Für jede Methode wird mithilfe der ersten Annotation die zugehörige HTTP-Anfragemethode definiert. Die darauffolgende `@RolesAllowed` Annotation wird für die Zugangssicherung benötigt. Dadurch wird in diesem Beispiel festgelegt, dass ein Nutzer eingeloggt sein muss und die Rolle „benutzer“ besitzen muss, um alle seiner Kleidungsstücke abzufragen. Die Transaktion wird mittels der dritten Annotation festgelegt. Zusätzlich ergeben sich weitere Annotationen, die für Swagger-UI benötigt werden.

3.3 Swagger-UI/OpenAPI (Manuel Arling)

OpenAPI ist ein Standard, welches die Beschreibung einer Programmierschnittstelle festlegt. Es soll den Entwicklern ermöglichen eine klar definierte und dokumentierte Programmschnittstelle für weitere Entwickler zur Verfügung zu stellen. Es soll dabei die Funktionen einer Schnittstelle darstellen, ohne den Programmcode zu durchschauen.

Kommentiert [MA13]: <https://spec.openapis.org/oas/latest.html#>

Swagger-UI realisiert dabei die von OpenAPI definierten Standards und ermöglicht innerhalb dieser Anwendung mittels geeigneter Annotationen, wie `@Operation(summary, description)`, `@ApiResponse`, `@Content` und `@Schema` den Nutzer eine klare und übersichtliche Struktur über alle möglichen Funktionen. Die hier aufgezählten Annotationen werden innerhalb dieser Anwendung angewendet.

Swagger-UI stellt eine grafische Plattform zur Verfügung, in der die Dokumentation visuell ansprechend angezeigt und auch mit der API interagiert werden kann. Diese lässt sich unter dem Pfad (<http://localhost:8080/virtuellerkleiderschrank/v1/q/swagger-ui/>) einsehen.

3.3.1 Daten Transfer Objekte (Manuel Arling)

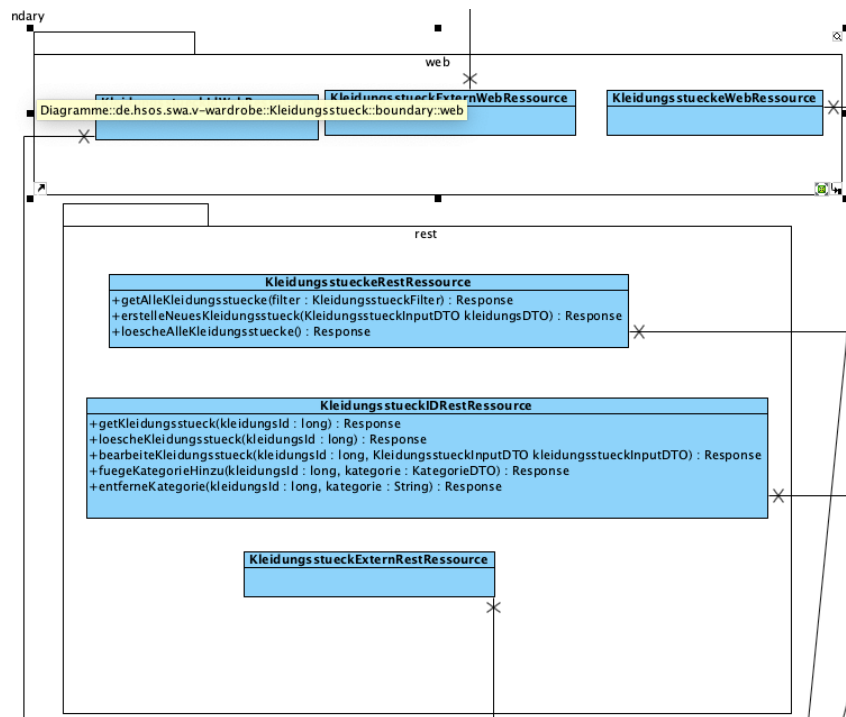
Daten Transfer Objekte dient für den Austausch von Daten zwischen zwei Systemen. Bei dieser API werden verschiedene DTOs für die jeweiligen Entitäten verwendet. DTOs legen dabei die Datenstruktur für eine Anfrage oder Antwort fest und formatieren so Daten, damit diese für die weiteren Schicht verwendet werden können.

Für Anfrage- und Antwort-Daten werden unterschiedliche DTOs verwendet. Dadurch lässt sich z.B. die Vergabe der ID bei einer neuen Entität durch den Server generieren, wodurch Kollisionen verhindert werden. So enthält ein Ausgangs-DTO z.B. zusätzlich zum Eingehenden-DTO das Feld `id`.

DTOs ermöglichen eine losere Kopplung zwischen der Entity-Schicht und den externen Systemen. So haben Änderungen an einer Entität, z.B. durch das Ergänzen von Attributen für die Datenbankverwaltung, keinen Einfluss auf externe Schichten.

3.3.2 Rest (Manuel Arling)

Das Rest-Package innerhalb der Boundary stellt die Maschine-Maschine-Schnittstelle dar. Dabei wird JSON als Medientyp für den Inhalt von Anfragen und Antworten verwendet.



Kommentiert [MA14]: Schauen ob im Text Referenzen auf snippet passen z.B. dritte Annotation

Abbildung 4 Ressource-Gruppierung und Klassendarstellung

Abbildung 4 zeigt die erstellten Ressourcen für die Umsetzung der in Kapitel 2 festgehaltenen Funktionalitäten. Jede Ressource Klasse besitzt durch die `@Path({pathname})` Annotation seine eigene URI. Die beiden Subdomänen besitzen dabei eine Ressource für das Holen und Löschen aller Entitäten der Subdomäne als DTO, sowie das Anlegen einer neuen Entität. Die zweite Ressource stellt dabei die `{Subdomäne}IDRestResource` dar. Diese beinhaltet alle Funktionalitäten, die für eine spezifische Entität zur Verfügung stehen. Die Subdomäne Kleidungsstücke besitzt noch die `KleidungsstueckExternRestResource`, welche einzig die Funktionalität fürs Erstellen eines neuen Kleidungsstücks per externen API zur Verfügung stellt. Outfits hingegen besitzt die `OutfitSharedRestResource`. Diese dient dazu, dass ein als geteiltes Outfit mithilfe des Pfades abgefragt werden kann.

Zusätzlich besitzen die Rest-Schnittstellen eine Implementierung des HATEOAS-Konzepts, welches in Kapitel x Grundlagen erklärt ist. Jedes Output DTO besitzt dabei das Attribut `links`, welches eine Liste von Links hält. Diese Links werden am Ende der

Ressource-Methode erstellt und in der Liste *links* des DTOs gespeichert. Damit die Antwort nicht mit zu vielen Informationen eines Links überfüllt wird, findet eine manuelle Serialisierung und Deserialisierung statt, welche in den Klassen *LinkSerializer* und *LinkDeserializer* umgesetzt sind und durch die entsprechende Annotation an der Variable des DTOs gesetzt wird.

3.3.3 Web (Manuel Arling)

Die Realisierung der Frontend-Anwendung Web-API wird in dem Web-Package umgesetzt. Diese stellt die Mensch-Maschine-Schnittstelle dar. Die Ressourcen im Web-Package der jeweiligen Subdomäne entsprechen den Ressourcen im Rest-Package, siehe Abschnitt x. Da die Schnittstelle für Webanwendungen spezialisiert ist, besitzen die Antworten der Ressourcen jedoch den Medientyp HTML. Denn so können von der Api erstellte Webseiten zur Verfügung gestellt werden, die im Browser eines Endgerätes dargestellt werden kann. Die Ressource-Methoden verwenden für den Inhalt unterschiedliche Medientypen. Zum Teil wird der Medientyp *application/x-www-form-urlencoded* für das Verarbeiten von einfachen Formularen. Dazu zählen die Formulare fürs Erstellen eines neuen Kleidungsstücks und eines Outfits. Für das Bearbeiten eines Kleidungsstückes oder eines Outfits wird hingegen ein komplexeres Formular benötigt, auf welches in Kapitel x eingegangen wird. Das Formular hingegen verwendet den Medientyp *application/json* innerhalb einer PUT-Anfrage. Deswegen ist der Endpunkt in der API entsprechend den von der Webseite verwendeten Eigenschaften implementiert, da das BFF-Pattern umgesetzt wird.

Zusätzlich wird für das Bearbeiten einer Entität keine PATCH-Anfrage ausgeführt, sondern eine PUT-Anfrage, da in der Web-Ansicht der Nutzer immer das komplette Form angezeigt bekommt. Würde die PATCH-Anfrage verwendet werden, müsste per JavaScript Funktion manuell geprüft werden, welche Daten geändert wurden. Dies stellt für diese Anwendung jedoch keinen großen Mehrwert dar, weshalb eine PUT-Anfrage durchgeführt wird.

Für die Web-Schnittstelle werden hauptsächlich HTML-Seiten als Antwort verschickt. Teilweise wird jedoch auch eine Antwort mit einem Redirect (303, seeOther) versehen. Dies findet z.B. bei der Erstellung einer neuen Entität statt. Denn so wird der Benutzer nach dem erfolgreichen Erstellen direkt zur Detailansicht weitergeleitet, dadurch muss der Benutzer nicht manuell nach dem Erstellen auf das Element klicken, um weitere Informationen einzutragen. Zusätzlich setzt es damit das Post/Redirect/Get-Pattern, welches dafür dient mehrfaches Senden eines Post zu verhindern.

Kommentiert [MA15]: <https://www.theserverside.com/news/1365146/Redirect-After-Post>

3.4 Control (Manuel Arling)

Die Control-Schicht realisiert die Geschäftslogik der API. Dazu gehört die Regelung der Datenverarbeitung. Deswegen bestimmt die Controlschicht welche Funktionen in der Entitäts-Schicht aufgerufen werden. Da jedoch keine Berechnungen benötigt sind und Regelungen bezüglich der Zugriffssteuerung schon durch Annotationen an den Boundary Ressourcen erfolgt, ist die Control-Schicht in dieser Anwendung kompakt. Dennoch validiert die Control-Schicht die übergebenen DTOs, mithilfe der von Annotation `@Valid`.

```
public class KleidungsstueckInputDTO {  
  
    @NotBlank(message="Groesse darf nicht leer sein")  
    public String groesse;  
    @NotNull(message="Farbe darf nicht leer sein und nur einer der guelti-  
gen Farben enthalten")  
    public Farbe farbe;  
    @NotNull(message="Typ darf nicht leer sein und nur einer der gueltigen  
Typen enthalten")  
    public Typ typ;  
    @NotBlank(message="HaendlerName darf nicht leer sein")  
    public String name;  
    public List<String> kategorien = new ArrayList<>();  
}
```

Abbildung 5 DTO: KleidungsstueckInputDTO

Die einzelnen DTOs besitzen dazu zusätzliche Annotationen an den zu prüfenden Attributen. Der Code Snippet x zeigt dabei die Vorgaben für die Validierung. Die Annotation `@NotBlank` prüft, ob die Variable nicht null ist und auch mindestens ein Zeichen enthält, welches kein Leerzeichen darstellt. Die Annotation `@NotNull` hingegen prüft lediglich, ob die Variable nicht null ist. Den Annotationen kann eine Nachricht mit dem Attribute `message` übergeben werden, welche bei einem Misserfolg der Validierung in die Antwort gesetzt wird.

Da für ein neues Kleidungsstück alle Variablen außer Kategorien benötigt wird, sind diese alle mit einer Überprüfung annotiert. Kategorien besitzt keine Überprüfung, da so der Nutzer auch ein Kleidungsstück erstellen kann, welches keine Kategorien besitzt.

3.5 Anti-Corruption Layer (Manuel Arling)

Eine Anti-Corruption-Layer (ACL) ist ein Entwurfsmuster. Es ermöglicht eine Einbindung von zusätzlichen Systemen. Dabei wird in diesem Kapitel eine interne ACL dargestellt, welche die Kommunikation zwischen den einzelnen Subdomänen regelt. Zusätzlich wird eine externe ACL für die Anbindung einer externen API verwendet und ebenfalls erklärt. Die Verwendung einer ACL ermöglicht es eine lose Kopplung zwischen den jeweiligen Systemen zu realisieren. Durch die interne ACL können somit Kleidungsstücke und Outfits unabhängig voneinander entwickelt und weiterentwickelt werden. Beide ACLs verwenden eine Kombination des Facade und Adapter Patterns. Das Facade-Pattern stellt vereinfachte Schnittstelle dar, welche die interne Anwendung für das Aufrufen eines externen Systems verwendet. Das Adapter-Pattern dient dabei zur Übersetzung der Aufrufe der internen Anwendung auf die Aufrufe der externen API. Zusätzlich übersetzt es die erhaltenen Daten in ein für die interne Anwendung gewünschtes Format. Dadurch muss die interne Anwendung keinerlei Detailinformationen zu den externen Systemen besitzen.

3.5.1 Interne ACL (Thomas Meese)

Für die Darstellung von Outfits im System eigenen Frontend werden Informationen zu den jeweiligen Kleidungsstücken benötigt. Dies ist nötig, da die Repräsentation der Kleidungsstücke innerhalb der Outfits über die hinterlegten *KleidungsIds* erfolgt. Die Informationen werden über eine interne ACL bereitgestellt, welche im Modul Kleidungsstücke zu verorten ist. Durch die Verwendung der ACL kann eine lose Kopplung trotz vorliegendem Cross-Cutting-Concern erreicht werden.

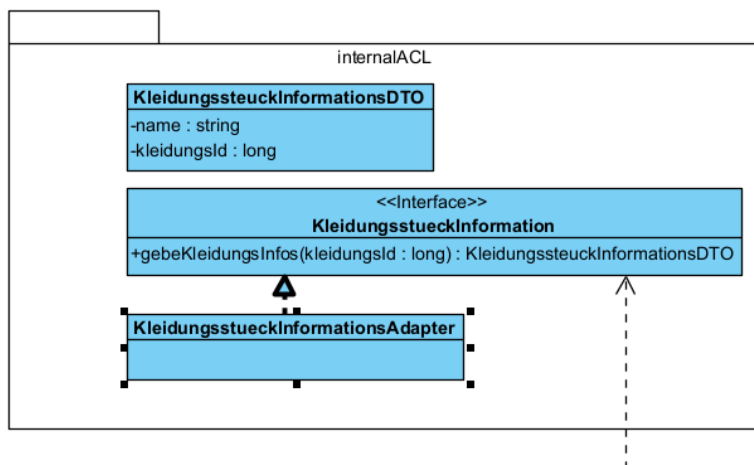


Abbildung 6 interne ACL

In Abbildung 4 ist der Aufbau der ACL erkennbar. Die benötigten Kleidungsinformationen werden durch den Adapter bereitgestellt und können durch die Facade (*KleidungsstückInformation*) vom Modul Outfit abgerufen werden. Die Informationen sind dabei in einem Separatem DTO enthalten.

3.5.2 Externe ACL (Manuel Arling)

Da diese Anwendung den Benutzer die Möglichkeit bietet Kleidungsstücke lediglich durch die Angabe der Artikelnummer, der Größe und des Händlers zum virtuellen Kleiderschrank hinzuzufügen, muss die Anwendung für jeden unterstützten Händler auf die zugehörige externe API zugreifen. Unterstützt wird der Händler H&M, welche durch die API <https://rapidapi.com/apidojo/api/hm-hennes-mauritz> möglich ist. Die API stellt dabei unterschiedliche Möglichkeiten zur Verfügung. Für die definierte Anforderung ist es von Nöten allein durch die Artikelnummer Informationen zu dem gesuchten Kleidungsstück zu erhalten. Die verwendete externe API bietet dabei den Endpoint „/products/detail“ zur Verfügung. Durch die Angabe der benötigten Query Parameter *lang*, *country* und *productcode* erhält man ein JSON-Objekt, welches im Attribut *product* mehr als 40 Attribute zu dem gefundenen Produkt enthält. Diese enthalten die für diese Anwendung benötigten Informationen.

Damit die Verwendung der externen API möglich ist, wird eine externe ACL aufgebaut. Diese isoliert die Integration der API in das System und beschäftigt sich mit den Anfragen und dem Übersetzen der unterschiedlichen Datenstrukturen beider APIs. Da die Anwendung die Anbindung von mehreren Händlern ermöglichen soll, wird das Facade-Pattern umgesetzt und jede externe API wird durch einen Adapter eingebunden.

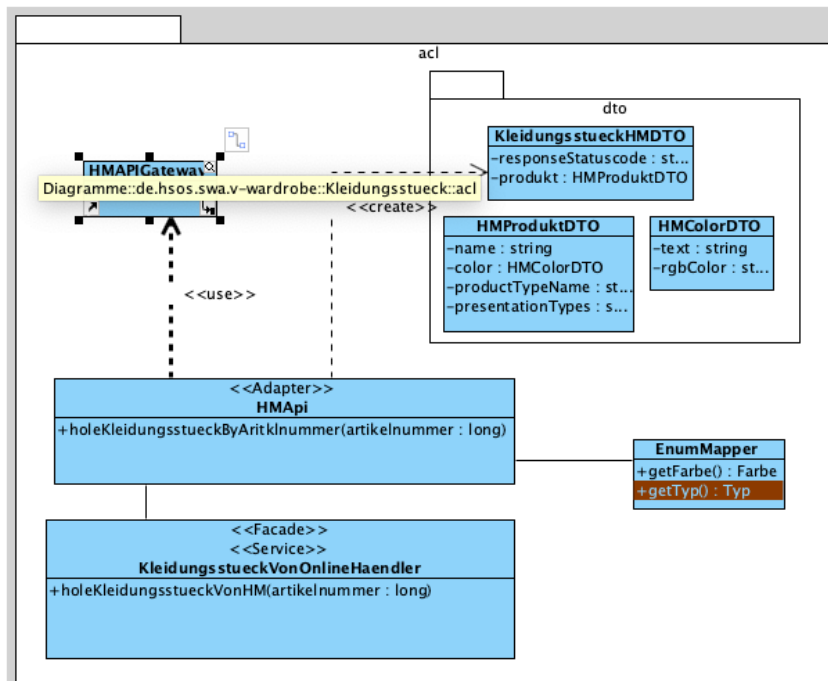


Abbildung 7 ACL für externe APIs

In Abbildung 7 ist der Aufbau der externen ACL zu erkennen. Die Facade ermöglicht der internen Anwendung, die Verwendung einer Schnittstelle für unterschiedliche Händler. Jede Methode der Facade stellt dabei die Anbindung einer externen API dar (hier H&M). Die Facade ruft daraufhin den zugehörigen Adapter der externen API auf. Die ACL lässt sich durch weitere externe APIs erweitern, dafür ist lediglich ein Adapter für die weitere API zu implementieren, sowie die Facade um eine Methode für den Anbieter zu erweitern, welche den erstellten Adapter verwendet. Für das Übersetzen der von der H&M erhaltenen Daten werden weitere DTOs benötigt, sowie ein Mapping der neuen DTOs auf die für das interne benötigte DTO. Dafür sind folgende DTOs verwendet:

- KleidungsstueckHMDTO: stellt das DTO für die Antwort der externen API dar.
- HMProduktDTO: ist in der KleidungsstueckHMDTO als Variable enthalten. Dieses DTO beinhaltet die für interne Anwendung benötigten Informationen.
- HMCOLORDTO: enthält die Informationen bezüglich der Farbe des HM Produkts.

```
public class HMProduktDTO {
    public String name;
```

```
public HMColorDTO color;  
public String productTypeName;  
public String presentationTypes;  
}
```

Der Code Snippet x zeigt den allgemeinen Aufbau des HMPProductDTOs. Es enthält zum einen den Namen des Produktes, zusätzlich die Farbe des Typen HMColorDTO, welches wiederum die Farbe als Text und als RGB-Farbe zur Verfügung stellt. Für den Typen werden die beiden Felder *productTypeName* und *presentationTypes* zur Verfügung gestellt. Die Verwendung beider Felder ermöglicht ein erfolgreicher Mappen des HM-Typen auf die von der internen verwendeten Anwendung, da teilweise nur einer der beiden Felder gesetzt oder jedoch einer der beiden Felder beispielhaft einen anderen Namen für den gleichen Typen verwendet.

Da die interne Anwendung für die Felder *farbe* und *typ* zwei vordefinierte Enums verwendet, muss ein sicheres Mapping zwischen den unterschiedlichen Typen der beiden DTOs erfolgen.

Dies ermöglicht die Klasse *EnumMapper*, welches die Übersetzung übernimmt. Dazu stellt der Mapper für jedes Enum eine Methode zur Verfügung, die einzugehöriges Objekt des jeweiligen Enums zurpückgibt.

3.6 Authentifizierung der Nutzer (Thomas Meese)

Um die umgesetzte API zu schützen, wird eine Zugriffskontrolle mittels OIDC und Keycloak umgesetzt. Im Kontext von OAuth gewährt hier der Nutzer Anwendungen (wie z.B. dem Frontend) eingeschränkten Zugriff auf seine geschützte Ressource (Die Kleidungsstücke bzw. Outfits, die dem Nutzer gehören). Zusätzlich erfolgt eine rollenbasierte Zugriffskontrolle der Nutzer (RBAC). Dabei wird geprüft, ob es sich bei dem authentifizierten Nutzer tatsächlich um einen Benutzer des Systems handelt. Generell ist Authentifizierung bei allen Funktionen nötig die eine Interaktion mit Objekten erfordern, die zu einem spezifischen Nutzer gehören, da sichergestellt werden muss, dass nur dieser Zugriff auf diese hat. Eine Ausnahme hierbei sind die geteilten Outfits, da hier eine „weitergeben“ an andere explizit erlaubt wurde. Neben dem Zugriff auf geteilte Outfits haben nicht registrierte Nutzer nur die Möglichkeit sich anzumelden oder ein Konto zu erstellen.

3.6.1 Umsetzung von Keycloak

Keycloak kann nach dem einbinden der Extension in der Pom.xml durch die Quarkus Konfig-Datei eingerichtet werden. Hierbei wird für diese API eine hybride Authentication-Routine verwendet, da ein Zugriff sowohl mit Swagger-UI also auch dem eigenen Frontend erwünscht ist. Als Grant Type wird hier Authorization Code kurz „code“ verwendet. Da Keycloak die Möglichkeit bietet einige Konfigurationen in einer eigenen UI vorzunehmen wird hierfür der Port 8180 konfiguriert. Ebenfalls wird ein Default-Benutzer konfiguriert (Gustav), der mit den Anmeldedaten (Nutzername: gustav, Passwort: einfach) verwendet werden kann. Über die Ui gibt es beispielsweise die Option eine selbstständige Registrierung der Nutzer mit automatischer Rollenzuweisung zu ermöglichen, wodurch sich eigene Forms bzw. Webseiten hierfür erübrigen. Damit die Konfiguration auch bei einem Serverneustart erhalten bleibt, kann die Konfiguration über die Ui als

JSON exportiert und in der Quarkus Konfig-Datei gesetzt und eingelesen werden. Jedoch funktioniert das Einlesen der Datei zumindest bei diesem System nur fehlerhaft. Einige der gesetzten Attribute, wie das Aktivieren der Registrierung werden nicht übernommen. Aus diesem Grund wird für das Anlegen neuer Benutzer eine andere Lösung verwendet.

3.6.2 Workaround mit dem Keycloak Admin-Client

Neben dem Konfigurieren und Verwalten von Realms mittels der UI bietet Keycloak hierfür auch eine programmatische Lösung mit dem sog. Admin-Client, der in ein Java-Projekt eingebunden werden kann. Durch die Einbindung kann die Verwaltung auch über Java-Methoden erfolgen. In diesem System wird so das Anlegen neuer Benutzer ermöglicht. Hierfür wird eine separate Subdomäne Benutzer angelegt. Die Kommunikation mit der Keycloak-API findet über das Gateway statt. Da Keycloak über eine eigene Datenbank für das Persistieren der Nutzer verfügt, muss kein weiteres Repository verwendet werden, wie es zum Beispiel bei http-basic der Fall wäre.

```
public boolean legeBenutzerkontoAn(NutzerDTO nutzerdaten) {  
    //Nutzer erstellen  
    UserRepresentation nutzer = new UserRepresentation();  
    nutzer.setUsername(nutzerdaten.benutzername);  
    nutzer.setEnabled(true);  
    nutzer.setFirstName(nutzerdaten.vorname);  
    nutzer.setLastName(nutzerdaten.nachname);  
    nutzer.setEmail(nutzerdaten.email);  
    try{  
        RealmResource realm = this.keycloak.realm(realmName);  
        UsersResource userResource = realm.users();  
        Response response = userResource.create(nutzer);  
        //Userid bestimmen  
        String userId =  
            response.getLocation().getPath().substring(response.getLocation()  
                .getPath().lastIndexOf("/") +1);  
        //Benutzer Rolle zuweisen  
        RoleRepresentation role =  
            realm.roles().get(roleName).toRepresentation();  
        userResource.get(userId).roles().realmLevel()  
            .add(Arrays.asList(role));  
        //Credentials generieren  
        CredentialRepresentation credential =  
            new CredentialRepresentation();  
        credential.setTemporary(false);  
        credential.setType(CredentialRepresentation.PASSWORD);  
        credential.setValue(nutzerdaten.passwort);  
        //Credential Setzen  
        userResource.get(userId).resetPassword(credential);  
        return true;  
    }catch(Exception exception){  
        return false;  
    }  
}
```

Snippet 4 Methode: Lege Benutzerkonto an

In Snippet 4 ist die Umsetzung der Funktion innerhalb des Gateways zum Anlegen eines neuen Benutzers erkennbar. Für die Interaktion mit Keycloak werden API-Zugriffe verwendet, die durch den Admin-Client möglich sind. Es wird ein neuer Nutzer erzeugt und im Quarkus-Realm erstellt. Anschließend wird dem Nutzer die Rolle Benutzer zugewiesen und das von ihm gewählte Passwort gesetzt. Bei der Umsetzung hat sich gezeigt, dass die Verwendung von JSONB zumindest bei diesem System zu Fehlern bei der Ausführung führt. Hier schlagen die Anfragen fortwährend mit dem Fehler „auth token is null“ fehl. Unter der Verwendung von JACKSON statt JSONB tritt der Fehler nicht mehr auf. Die API selbst wirft auch Fehler bei falschen Nutzerdaten, da beispielsweise der Benutzername eindeutig sein muss, was bei der Verwendung der API zu beachten ist.

Das Anlegen von Benutzer kann in diesem System mittels Post-Request an die API oder über ein Formular im Frontend erfolgen.

3.7 Entitys (Thomas Meese)

Ein Entity ist ein Objekt was durch seine Identität und nicht durch seine Attribute definiert wird. Im der umgesetzten API gibt es zwei Entitys. Das erste dieser beiden, das Kleidungsstück, ist das Hauptobjekt. Es beschreibt die Objekte, die im Kleiderschrank „gelagert“ werden können. Dabei besteht ein Kleidungsstück aus einem Namen, der Farbe, dem Typ (z.B. Pullover), der Größe, der Kategorie (z.B. Sommer oder Freizeit) und dem Benutzer, zu dem es gehört. Farbe und Typ werden durch Enums vordefiniert. Die Kategorien sind vollständig frei vom Nutzer wählbar. Das Outfit-Entity benötigt die Kleidungsstücke, da diese aus Ihnen zusammengestellt werden kann. Neben den Kleidungsstücken die Id-Referenz vorliegen besteht ein Outfit noch aus den Attributen aus dem Namen des Outfits, den Kategorien, dem Status (geteilt oder nicht) und dem Benutzer, zu dem es gehört. Auch hier sind die Kategorien frei wählbar. Da die Entitys in einer Datenbank gespeichert werden benötigen sie Ids um sie eindeutig identifizieren zu können. Diese werden durch Hibernate-Orm mittels der Annotation `@GeneratedValue` automatisch generiert.

Kommentiert [TM16]: Quelle finden

3.8 Gateways (Thomas Meese)

Die Gateways dienen der Kommunikation mit der Datenbank. Hier sind alle Funktionen zu verorten die sich auf die Interaktion mit den persistenten Entitys bezieht. Für die Interaktion mit der Datenbank wird der Entitymanager von Javax Persistence verwendet. Der lesende Zugriff erfolgt hierbei über Querrys, die via Typedquery ausgeführt werden. Für den schreibenden Zugriff werden die Methoden `persist()` und `remove()` verwendet. Hierbei ist zu beachten das korrekte Entitys übergeben werden müssen, da es ansonsten zu Fehlern kommt. Bei den Methoden, die einen schreibenden Zugriff auf die Datenbank ermöglichen müssen mit einem `@Transactional` annotiert werden. Sollte es bei den Methoden zu Fehler kommen erfolgt die Ausnahmebehandlung in der Methode in der sie Auftritt, das heißt Exceptions werden hier nicht nach außen weitergegeben.

3.8.1 Events

Da das Entity Kleidungsstück auch in den Outfits enthalten ist gibt es bei den Crud-Funktionalitäten etwas zu beachten. Wenn ein Nutzer ein Kleidungsstück löscht, muss auch die Id Referenz aus jedem Outfit entfernt werden, indem es vorkommt. Hierfür werden die sogenannten Events verwendet. Insgesamt werden zwei Events umgesetzt. Das erste Event behandelt den Fall, dass ein Kleidungsstück gelöscht wurde, während das zweite aktiv wird, wenn alle Kleidungsstücke vom Nutzer gelöscht wurden. Sie laufen komplett in den Repositories ab. Dies hat zwei Hauptgründe, zum einen handelt es sich dabei um eine direkte Bearbeitung der Entitys, weswegen die Behandlung im Repository stattfinden muss. Zum anderen erfolgt, wie bereits erwähnt, die Fehlerbehandlung dort wo der Fehler auftritt. Um auf Fehler beim Löschen korrekt reagieren zu können muss das Starten der Events auch im Repository geschehen. Dabei feuert das Kleidungsstück-Repository das jeweilige Event, wenn ein Löschereignis auftritt. Wenn ein Kleidungsstück entfernt wird oder ein Fehler beim Löschen aller Kleidungsstücke auftritt wird das erste Event gefeuert. Hierdurch kann sichergestellt werden dass diejenigen Kleidungsstücke, die beim Löschen aller Kleidungsstücke tatsächlich entfernt wurden, bevor der Fehler auftritt, auch aus den Outfits entfernt werden. Bei der Behandlung der Events erfolgt dabei im Outfit-Repository. Im ersten Fall werden Alle Outfits geprüft, ob das gelöschte Kleidungsstück darin vorkommt, wenn das der Fall ist, wird es entfernt. Das zweite Event hingegen wird nur gefeuert, wenn tatsächlich alle Kleidungsstücke gelöscht wurden. Die Behandlung des Events ist trivial, da hier einfach bei allen Outfits alle Kleidungsstücke entfernt werden können und sich somit eine Iteration über die Jeweiligen Kleidungslisten erübrigt.

3.9 Ausfallsicherheit (Thomas Meese)

Um die Verfügbarkeit der umgesetzten API zu verbessern, werden die Enterprise Fault-tolerance Annotationen verwendet. Diese sind an kritischen Stellen der API einzusetzen. Eine dieser Stellen ist die Verwendung einer Externen API, die über einen Rest-Client eingebunden wird. Bei Zugriffen auf externe Systeme kann es vorkommen dass diese temporär nicht erreichbar oder überlastet sind, wodurch es zu Fehlern bei den Anfragen kommen kann. Aus diesem Grund ist das Gateway mit einem `@Retry` annotiert. Hierdurch wird die Anfrage bis zu vier wiederholt, falls es dabei zu falschen Ergebnissen kommt. Zusätzlich gibt es die Möglichkeit auftretende Fehler zu White- bzw. zu Blacklisten. Dies hat den Vorteil dass die wiederholte Anfrage nur dann ausgeführt wird, wenn ein serverseitiger Fehler auftritt. So macht es beispielsweise keinen Sinn eine wiederholte Anfrage auf eine Ressource zu tätigen die nicht existiert, da diese immer scheitern wird. Zusätzlich wird hier die Annotation `@CircuitBreaker` verwendet. Der Circuitbreaker stoppt dem Zugriff auf diese Methode, wenn die Hälfte der gehaltenen Ergebnisse Fehlschläge (Ratio ist auch anpassbar, Default ist 50%) sind. Nach dem Ablauf einer gewissen konfigurierbaren Zeit werden Testanfragen getätigt, falls diese Erfolgreich sind wird die Methode wieder freigegeben. Dabei empfiehlt es sich eine Anzahl größer eins zu nehmen, da ein einzelnes positives Ergebnis auch ein Zufall sein könnte. Ein weiterer kritischer Punkt ist die Interaktion mit dem Nutzer, im speziellen, wenn davon auszugehen ist, dass dieser Ergebnisse einer Anfrage in kurzer Zeit erwartet und diese nur eine begrenzte Zeit gültig sind. Dies ist bei der umgesetzten Filterfunktion der Fall. Hier ist davon auszugehen, dass der Nutzer unmittelbar nach dem Absenden der Anfrage das

Kommentiert [TM17]: Besser umschreiben

Ergebnis erhalten möchte. Aus diesem Grund ist die jeweilige Methode mit einem `@Timeout` annotiert, was bewirkt, dass die Anfrage abgebrochen wird, wenn die Ausführung zu lange benötigt. Um dem Nutzer trotz Scheitern der Anfrage ein Ergebnis zu liefern, wird eine Fallback-Methode (`@Fallback` Annotation) umgesetzt, die eine leere Liste von Objekten (Kleidungsstücke oder Outfits) liefert. Somit erhält der Nutzer bei Scheitern der Anfrage die Antwort, dass keine entsprechenden Objekte gefunden wurden.

3.10 Frontend

Das Frontend für das System wird mittels der im Kapitel x dargestellten Template Engine Qute umgesetzt und durch JavaScript Funktionen sowie dem Framework Bootstrap erweitert.

3.10.1 Generelle Ansicht (Thomas Meese)

Abbildung 8 Hauptansicht Meine Kleidungsstücke

In Abbildung 5 ist eine der beiden Hauptseiten exemplarisch zu sehen. Der Benutzer kann die beiden Seiten über das Menü erreichen. In der Navigationsleiste befindet sich ebenfalls ein Abmeldebutton, um sich vom System zu trennen. Beide Hauptseiten haben den gleichen generellen Aufbau. Es werden beim Laden alle vorhandenen Objekte (Kleidungsstücke oder Outfits) in einer Tabelle angezeigt. Die Ergebnisse können mittels des Filters angepasst werden. Dabei kann je ein Schlüsselwort pro Filterkategorie vom Nutzer eingetragen werden. Der Filter unterscheidet sich bei den beiden Hauptseiten leicht, bei der Outfit-Seite können die Ergebnisse nur mit Namen und Kategorie gefiltert werden. Der Nutzer hat die Möglichkeit neue Kleidungsstücke (oder Outfits je nach Seite) hinzufügen oder alle Kleidungsstücke (bzw. Outfits) auf einmal zu Löschen. Über die Aktionsbuttons können einzelne Elemente bearbeitet oder entfernt werden.

3.10.2 Forms und Detail Ansicht (Manuel)

In der generellen Ansicht der Kleidungsstücke kann der Benutzer durch das Klicken des Buttons „Neues Kleidungsstück erstellen“ ein neues Kleidungsstück erstellen, dabei wird ein Modal geöffnet, welches sowohl ein Formular für die manuelle Eingabe der Daten als auch ein Formular für die Verwendung einer Externen API.

The screenshot displays a web interface for managing clothing items. A modal window titled "Neues Kleidungsstück erstellen" is open, allowing the user to create a new item. The modal has two tabs: "Manuell" (Manually) and "Extern" (Externally). The "Manuell" tab is currently active, showing form fields for "Name", "Größe" (Size), "Typ" (Type), and "Farbe" (Color). The "Name" field has a placeholder text "Gib deinem Kleidungsstück einen eigenen Namen". The "Größe" field has a placeholder text "Du kannst alle möglichen Größen eingeben". The "Typ" and "Farbe" fields are dropdown menus with placeholder text "Wähle den Kleidungsstyp" and "Wähle die Hauptfarbe" respectively. A blue button "Kleidungsstück erstellen" is located at the bottom of the modal. In the background, a table titled "Meine Kleidungsstücke" is visible, showing columns for "Typ", "Farbe", and "Aktionen". The "Aktionen" column contains edit and delete icons for each item.

Abbildung 9 Modal zum Erstellen von Kleidungsstücken

Abbildung 9 zeigt dabei das Modal fürs Erstellen eines neuen Kleidungsstückes, welches das manuelle Formulare darstellt. Durch die beiden oberen Buttons hat der Nutzer die Möglichkeit schnell und ohne Aufruf einer anderen Seite zwischen den beiden Formularen (manuell und extern) zu wechseln. Indem der Button für das aktuell ausgewählte Formular blau hinterlegt ist, wird dem Benutzer visuell angezeigt, welches der beiden Formular momentan aktiv ist. Dadurch wird die User Experience erhöht und es kommt zu keiner Unklarheit für die Verwendung der jeweiligen Formulare. Die beiden Formulare beinhaltet dabei lediglich die für das Formular benötigten Pflichtfelder. Dadurch wird der Benutzer nicht durch für Ihn unnötigen Informationen konfrontiert. Die generelle Ansicht der Outfits ist ebenfalls nach dem Prinzip aufgebaut, dass durch das Klicken des Buttons

„Neues Outfit erstellen“ das zugehörige Formular in dem geöffneten Modal angezeigt wird. Jedoch wird hier nur ein Modal zur Verfügung gestellt. Dieses Formular beinhaltet ebenfalls nur die notwendigen Informationen für das Erstellen eines neuen Outfits, nämlich die Vergabe des Namens.

Nach dem Absenden des Forms, mittels des auf dem unteren Teil dargestellten Buttons, wird der Benutzer bei einer erfolgreichen Erstellung des Objekts zu der Detailansicht des Objekts umgeleitet.

Menü

Name

Sport

Kategorien

Sport Freizeit

Kleidungsstuecke:

Name des Kleidungsstücks
Sport T-Shirt Nike
Kurzhose Sporthose Nike
Jogginghose Nike
Muskelshirt

Outfit bearbeiten Outfit teilen

Abbildung 11 Detailansicht Outfit

Die Detailansicht des Kleidungsstücks und des Outfits dienen dafür dem Nutzer jegliche Informationen zu dem Objekt zu präsentieren, wie es in Abbildung 11 zu erkennen ist. Die jeweilige Detailansicht verwendet dabei das fürs Erstellen verwendete Form, wobei diese um optionale Informationen ergänzt sind, wodurch der Nutzer ab diesem Zeitpunkt die Möglichkeiten hat optionale Informationen zu ergänzen. Jedoch werden jegliche Felder zuerst nur für den lesenden Zugriff dargestellt, dies wird dem Benutzer visuell durch das Ausgrauen eines Inputfeldes (hier Name) dargestellt. Der Benutzer besitzt auch nicht die Möglichkeit die Inputfelder anzuklicken, um den Text anzupassen. Um einzelne Informationen zu bearbeiten, muss der bearbeiten Button geklickt werden.

Menü

Name

Sport

☐ Outfit öffentlich

Kategorien

SportFreizeit

Kategorienamen...

Kategorie hinzufügen

Eine Kategorie kann z.B. ein Anlass oder eine Jahreszeit sein.

Kleidungsstücke:

Name des Kleidungsstücks	Aktionen
Sport T-Shirt Nike	
Kurzhose Sporthose Nike	
Jogginghose Nike	
Muskelshirt	

Änderungen speichern

Abbildung 12 Bearbeitungsansicht Outfit

Danach werden die zuvor nur für den lesenden Zugriff zur Verfügung gestellten Felder auch fürs Bearbeiten freigeschaltet. Ebenfalls werden zusätzliche Informationen und Buttons eingeblendet, was gut in der Abbildung 12 zu sehen ist. Eins dafür ist das Inputfeld fürs Eingeben einer neuen Kategorie, welches mit dem zum Input rechtsstehenden Button zu der obigen Liste hinzugefügt werden kann. Der in der Abbildung 12 vorhandene Plus Button ermöglicht es ein neues Kleidungsstück zum Outfit zu ergänzen. Dabei wird ein Modal geöffnet, welches eine Liste aller vom Benutzer erstellten Kleidungsstücke darstellt. Durch das Auswählen eines Kleidungsstücks wird das Modal automatisch geschlossen und das neue Kleidungsstück taucht in der Tabelle auf. Durch das Speichern der Änderungen wird das Formular an den entsprechenden Endpunkt der API geschickt. Dabei werden die Daten nicht mittels des Medientypen *application/x-www-form-urlencoded* übermittelt. Dies hat den Hintergrund, dass ein Formular nur Daten aus Inputfeldern übermittelt, dadurch müsste jede Kategorie und jedes Kleidungsstück als Inputfeld dargestellt werden. Dies würde dem Benutzer jedoch einen falschen Eindruck suggerieren. Der Benutzer würde denken, dass z.B. bei einem Kleidungsstück der Name

geändert werden kann, was jedoch nicht gewollt ist. Um die Benutzerfreundlichkeit zu verbessern werden, wie in Abbildung 12 zu sehen, die Kategorien als List-Elemente und Kleidungsstücke als Tabellen-Elemente dargestellt. Dadurch erkennt der Benutzer direkt, dass er keine verwirrenden Änderungen an diesen Elementen vornehmen kann. Damit jedoch jetzt diese Elemente dennoch mit in dem PUT-Request übermittelt werden, muss der Medientyp *application/json* verwendet werden. Dafür wird eine JavaScript Funktion benötigt, welche die jeweiligen Informationen in einen JSON-String darstellt. Da das HTML-Element sowieso nur GET und POST-Anfragen erstellen kann, muss an dieser Stelle sowieso JavaScript verwendet werden um eine PUT Anfrage für das dargestellte Objekt zu erzeugen.

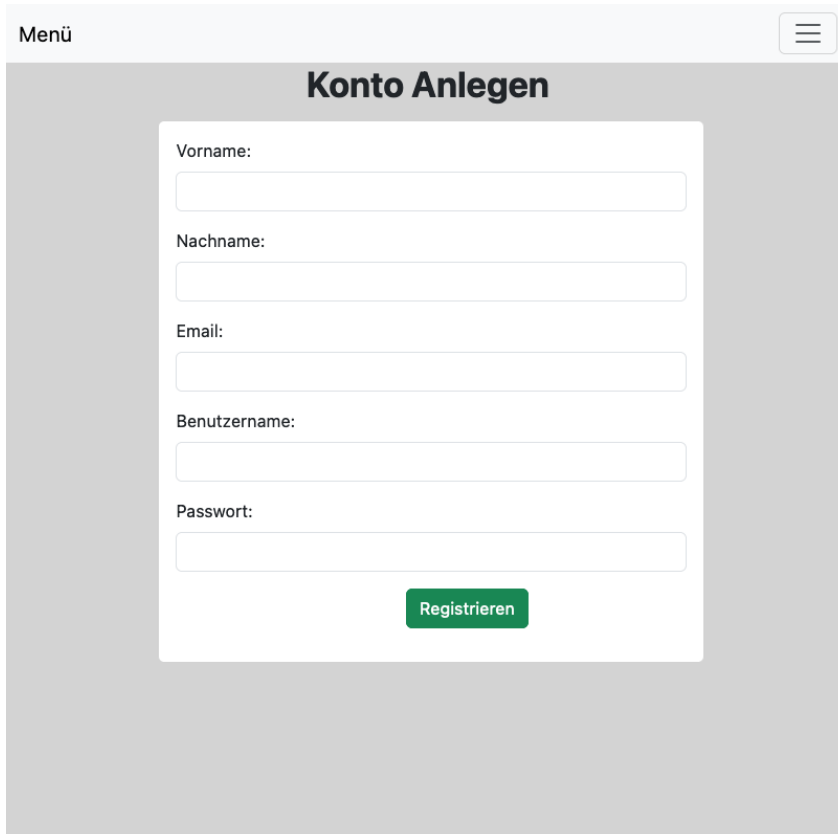
Requests die durch Javascript Code erfolgen werden mithilfe von *AJAX* oder *fetch* durchgeführt.

Kommentiert [MA18]: <https://wiki.selfhtml.org/wiki/HTML/Elemente/form>

Kommentiert [MA19]: Gucken was wir einheitlich nehmen sollten.

3.10.3 Direktlinkseiten (Thomas Meese)

Zwei Seiten dieser Anwendung dieser Anwendung sind nur über direktlinks erreichbar. Dabei handelt es sich um die geteilten Outfits und das Anlegen neuer Konten. Die Seite für ein geteiltes Outfit besitzt prinzipiell den gleichen Aufbau wie die zuvor beschriebene Detailansicht der Outfits. Der einzige Unterschied ist, dass hier alle Bearbeitungsfunktionen wegfallen, da dies nur dem Eigentümer des Outfits vorbehalten ist



The screenshot displays a web interface for creating an account. At the top left, there is a 'Menü' label. In the top right corner, there is a hamburger menu icon. The main heading is 'Konto Anlegen'. Below this, there is a white form box containing five input fields: 'Vorname:', 'Nachname:', 'Email:', 'Benutzername:', and 'Passwort:'. Each field is represented by a white rectangular input box. At the bottom of the form box is a green button with the text 'Registrieren'.

Abbildung 13 Konto anlegen

Abbildung 13 zeigt den Aufbau der Registerseite. Der Anwender hat die Möglichkeit die Angaben, welche für eine erfolgreiche Registrierung notwendig sind zu tätigen und anschließend zu bestätigen. Eine Validierung der einzelnen Formular Felder erfolgt dabei jedoch nicht. Nach dem sich der Anwender erfolgreich registriert hat kann er sich mit Benutzername und Passwort anmelden (sobald er eine der Hauptseiten aufgerufen hat) um Zugriff auf die restlichen Funktionen zu erhalten.

3.11 Qualität

Bei dem zu liefernden Produkt ist die Qualität der Software von hoher Priorität. Um diese bewerten zu können werden Qualitätsmerkmale benötigt.

3.11.1 Developer Experience und User Experience

Bei der Umsetzung der Anwendung spielt die anzustrebende Developer Experience eine wichtige Rolle. Durch das Erreichen des Level 3 im Richardson Maturity Model und der damit verbundenen Verwendung von HATEOAS wird dies nach außen hin erreicht. Rest-Clients haben die Möglichkeit anhand der Links durch die Anwendung zu navigieren, was eine separate Dokumentation nicht zwingend notwendig macht. Zusätzlich sind die Restschnittstellen mit Swagger-UI dokumentiert, was eine gute Visualisierung der Funktionen in Kombination mit den Anwendungsbeispielen ermöglicht.

Auch im Frontend wird Wert auf User Experience gelegt. Die einzelnen Ansichten der Anwendung sind responsiv gestaltet. Außerdem wird mit User Feedback gearbeitet, beispielsweise erscheint beim Bearbeiten eines Kleidungsstücks oder eines Outfits die neu hinzugefügte Kategorie sofort in der Liste. Auch beim Entfernen von Objekten wird der Nutzer zum einen darüber informiert und zum anderen muss das Löschen von ihm bestätigt werden, um ein versehentliches Löschen ausschließen zu können.

3.11.2 Korrektheit

Um die Korrektheit der Anwendung sicherzustellen, werden drei Verfahren verwendet. Das Frontend selbst kann verwendet werden, um Fehler in der Funktionsweise ausfindig zu machen und eine korrekte Funktionsweise zu gewährleisten. Ebenfalls erfolgt ein Testen der Funktionen der Anwendung mit Insomnia einem Tool, welches für das Designen und Testen von APIs entworfen ist. Als drittes Verfahren werden Rest Assured Test umgesetzt welches ein programmatisches Testen der Anwendung ermöglicht.

3.11.3 Sicherheit

Die Ressourcen der Anwender werden über die Rollenbasierte Zugriffskontrollen abgesichert. Die Umsetzung dieser erfolgt mittels Keycloak welches die aktuellen Protokolle OAuth2 und OIDC umsetzt.

3.11.4 Resilienz

Um die Verfügbarkeit der Anwendung zu verbessern, werden die Annotationen der Faultolerance verwendet. Hierdurch können kritische Stellen innerhalb der Anwendung wie Schnittstellen nach außen z.B. über den Rest-Client, welche ausfallen und somit die Funktionalität der Anwendung einschränken können, abgesichert. Auch zeitkritische Zugriffe wie das Filtern von Daten wird dadurch positiv beeinflusst.

4 Zusammenfassung und Fazit

Das Projekt hat das Ziel, eine API mithilfe des Quarkus-Frameworks zu entwickeln und zu implementieren, die als virtueller Kleiderschrank fungiert und es den Nutzern ermöglicht, ihre Kleidung digital zu verwalten. Dieses Ziel wurde durch die einzelnen Kapitel umgesetzt. Die erstellte API bietet Funktionen wie das Hinzufügen, Bearbeiten und Entfernen von Kleidungsstücken sowie das Erstellen und Verwalten von Outfits. Zusätzlich soll mindestens eine externe API über einen Rest-Client eingebunden werden, um es dem Nutzer zu ermöglichen, automatisch Kleidungsstücke aus Online-Shops hinzuzufügen. Der Nutzer kann seine erstellten Outfits teilen und anderen Nutzern lesenden Zugriff darauf ermöglichen, um beispielsweise für Kleidungsstücke eines Händlers zu werben. Ziel des Projekts ist es, dem Nutzer die beiden Schnittstellen Rest-Anwendung und Web-Anwendung zur Verfügung zu stellen. Insgesamt ist das Projekt vielversprechend und bietet eine interessante Möglichkeit, die Verwaltung von Kleidung zu vereinfachen und zu automatisieren. Es sind alle gewünschten Anforderungen umgesetzt worden. Es wurden ebenfalls mehrere Frameworks verwendet. Die Verwendung von den in Kapitel 2 dargestellten Patterns und Stile ermöglicht der API eine geeignete Softwarearchitektur.

4.1 Fazit

Mit dem Abschließen des Projekts wird eine API geliefert, die die gewünschten Funktionen umsetzt. Durch die Einhaltung der Qualitätsmerkmale weist die Restful API ebenfalls eine gute Qualität auf. Die API kann durch weitere Funktionen, wie das hochladen von Fotos, weitere Informationen oder die Unterstützung weiterer externer APIs/Händler verbessert werden.

5 Referenzen

- [@OAu] The OAuth 2.0 Authorization Framework, https://openid.net/specs/openid-connect-core-1_0-final.html
- [@OID] Open ID Connect Core 1.0, https://openid.net/specs/openid-connect-core-1_0-final.html
- [@MIC] Microservices werden zur Standardarchitektur für Anwendungen-ist es an der Zeit einzusteigen? <https://www.scaleway.com/de/blog/microservices-werden-zur-standardarchitektur-fur-anwendungen-ist-es-an-der-zeit-einzusteigen-cloud-trends-2022/>
- [@QUA] What is Quarkus?, <https://quarkus.io/about/>
- [@TEM] Template Engine - Was ist das?, <https://onlinemedien.blog/2018/01/template-engine-was-ist-das/>
- [@QUT] Qute Reference Guide, <https://quarkus.io/guides/qute-reference#type-safe-templates>
- [@BOO] Was ist Bootstrap, <https://it-talents.de/it-wissen/bootstrap/>
- [@CDI] Introduction to Contexts and Dependency Injection, <https://quarkus.io/guides/cdi>
- [@BCE] Web Components, Boundary Control Entity and unidirectional Dataflow with redux, https://adambien.blog/roller/abien/entry/web_components_boundary_control_entity
- [@ARC] Boundary Control Entity Architecture Pattern, <https://vaclavkosar.com/software/Boundary-Control-Entity-Architecture-The-Pattern-to-Structure-Your-Classes>
- [@DDD] Einführung in die Konzepte von Domain Driven Design, <https://entwickler.de/ddd/einfuehrung-in-die-konzepte-von-domain-driven-design-001>
- [@RMM] Richardson Maturity Model, <https://martinfowler.com/articles/richardson-MaturityModel.html>
- [@HAT] HATEOAS: Das steckt hinter dem Akronym, <https://www.ionos.de/digitalguide/websites/web-entwicklung/hateoas-alle-informationen-zu-der-rest-eigenschaft/>
- [@OPE] OpenAPI Specification v.3.1.0, <https://spec.openapis.org/oas/latest.html#>

- [@RAP] Redirect After Post, <https://www.theserverside.com/news/1365146/Redirect-After-Post>

- [STPS21] Stian Thorgersen, Pedro Igor Silva, Keycloak – Identity and Access Management for Modern Applications, Packt Publishing ldt, Birmingham, 2021

Eidesstattliche Erklärung

Hiermit erkläre ich/ erklären wir an Eides statt, dass ich / wir die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt habe / haben. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche einzeln kenntlich gemacht. Es wurden keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

17.02.2023

.....

Ort, Datum

.....

Unterschrift

(bei Gruppenarbeit die Unterschriften sämtlicher Gruppenmitglieder)

Urheberrechtliche Einwilligungserklärung

Hiermit erkläre ich/ Hiermit erklären wir, dass ich/wir damit einverstanden bin/sind, dass meine/ unsere Arbeit zum Zwecke des Plagiatsschutzes bei der Fa. Ephorus BV bis zu 5 Jahren in einer Datenbank für die Hochschule Osnabrück archiviert werden kann. Diese Einwilligung kann jederzeit widerrufen werden.

17.02.2023

.....

Ort, Datum

.....

Unterschrift

(bei Gruppenarbeit die Unterschriften sämtlicher Gruppenmitglieder)

Hinweis: Die urheberrechtliche Einwilligungserklärung ist freiwillig.