HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Visualisierung von Interface Representation Patterns: Automatisierung und Werkzeugintegration

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2017

Autoren: Sebnem Kaslack, Nicolas Dipner
Betreuer: Prof. Dr. Olaf Zimmermann, Institut für Software (IFS)
Experte: Dr. Gerald Reif, Innovation Process Technology, Zug
Gegenleser: Prof. Stefan Keller, Institut für Software (IFS)

Abstract

Ein Autorenteam der HSR erarbeitet zur Zeit mit externen Kooperationspartnern eine Patternsprache für messagebasierte Application Programming Interfaces. Die Verfasser dieser Bachelorarbeit entwarfen während ihrer Studienarbeit 50 Visualisierungen für diese Patternsprache, die es Dritten ermöglicht, eigene APIs anhand von Pattern-Icons zu dokumentieren. Das Ziel dieser Bachelorarbeit war es, die Visualisierungen in Konzept und Umsetzung zu optimieren und einen Dokumentationsassistenten zur Gestaltung von Web-API-Beschreibungen basierend auf der Patternsprache zu entwickeln.

Mit dem Prototyp einer Web-Applikation ist es möglich, ein bestehendes Web-API im OpenAPI Specification Format einzulesen und zu analysieren. Der Assistent visualisiert die Struktur des APIs und sucht darin anhand von Eigenschaften, Begriffen und hinterlegten API-Referenzen nach verwendeten Patterns. Diese werden dem Anwender in einer navigierbaren Diagrammhierarchie angezeigt. Das Resultat der Strukturanalyse und Pattern-Suche kann durch den Anwender überarbeitet werden. Den Benutzern des Dokumentationsassistenten steht alternativ die Möglichkeit zur Verfügung, ein API von Grund auf neu zu gestalten und über zu hinterlegende URI-Referenzen in der Freitextbeschreibung des APIs schlüsselwortbasiert nach Patterns zu suchen.

Die visuelle Darstellung des API mit den Pattern-Zuordnungen ist plattformunabhängig und komplementär zu den bisher verwendeten textuellen Formaten. Sie kann exportiert werden und gibt damit Dritten die Möglichkeit, einen schnellen Überblick über ein API zu erhalten und APIs zu vergleichen. Mit Hilfe der Kurzbeschreibungen zu den Pattern-Icons sowie Referenzen in die Patternsprache kann sich ein Benutzer der Applikation über die verwendeten Patterns informieren, was den Einarbeitungsaufwand bei API-Verwendung und -Wartung senkt.

Management Summary

Ausgangslage

Das Akronym API steht für „Application Programming Interface". Ein API ermöglicht den Austausch von Daten bzw. die Kommunikation zwischen zwei Systemen. In der Webentwicklung gibt es verschiedenste derartiger Schnittstellen, die für den Gebrauch durch Nutzer beschrieben werden müssen. Die Dokumentationen solcher APIs sind oft zweckmässig aufgebaut und stark an das System gebunden, was die Einarbeitungszeit bei der API-Anwendung erhöht. Ein offizieller Standard für die Dokumentation solcher Schnittstellen besteht nicht, daher unterscheiden sich API-Beschreibungen stark voneinander. Dass solche Standardisierungen durchaus erwünscht wären, bestätigen Interessensgruppen wie beispielsweise die OpenAPI Initiative. Die OAI bemüht sich darum, einen Standard für Representational-State-Transfer-Schnittstellen - ein Architektur- bzw. Integrationsstil für verteilte Systeme - zu schaffen.

Die Recherecheergebnisse der Studienarbeit zeigen auf, dass sich solche Standards auch bei prominenten API-Anbieter nicht durchsetzen. Ein Grund dafür ist die Tatsache, dass die Nutzer dieser APIs zwangsläufig auf das anzubindende System angewiesen sind. Für den Schnittstellenanbieter ist die Standardisierung mit mehr Aufwand verbunden und nicht zwingend lohnenswert, wenn der Anwender ohnehin die Schnittstelle nutzt. Daher treffen Entwickler oft auf Schnittstellenbeschreibungen mit eigener Gliederung, meist in Freitext beschrieben.

Unter der Leitung von Herrn Zimmermann arbeitet ein Autorenteam an einer plattformübergreifenden Pattern-Sprache für Web APIs. Ein Pattern entsteht aus erfolgreichen Engineering-Projekten und beschreibt einen generischen Lösungsvorschlag, welcher ein immer wiederauftretendes Problem adressiert und sich in der Anwendung bewährt hat. Im Rahmen der Studienarbeit haben sich die Verfasser dieser Bachelorarbeit bereits mit der eben genannten Patternsprache auseinandergesetzt und Visualisierungen für die bestehenden Patterns entworfen. Anhand dieser Patterns soll ein Dokumentationsassistent entwickelt werden, der das beschriebene Problem der vielfältigen API-Beschreibungen adressiert.

Mit dem Assistenten sollen API-Beschreibungen eingelesen, Struktur und Inhalt analysiert und mit Hilfe der Pattern-Visualisierungen dargestellt werden. Die Absicht dahinter ist, dass der Anwender durch die generierte Darstellung ein API - gestützt durch die Pattern-Sprache - schneller versteht.

Vorgehen

Die Planung des Projektes folgte dem iterativen Vorgehen des Unified Process. Zur Entwicklung der Web-Applikation erfasste das Projektteam die Anforderungen und erstellte User Stories, die Szenarien aufzeigen, welche durch die Applikation abgedeckt werden sollen. Während der Evaluationsphase plante das Projektteam den Aufbau des Applikations-Prototypen, indem es Technologien evaluierte, grundlegende Architekturentscheidungen traf und Entwürfe für die Benutzeroberfläche gestaltete.

Das Projektteam entschloss sich in Folge des Evaluationsprozesses dafür, mit JHipster, einem Framework zur Erstellung von Webapplikationen, zu arbeiten. Diese Entscheidung führte dazu, dass der Aufwand für den Aufbau der grundlegen Applikationsstruktur abnahm, jedoch neue Risiken auftraten, da das Projektteam keine Erfahrung diesem Framework vorzuweisen hatte. Um effizient zu entwickeln, wurde die Programmierarbeit auf zwei Bereiche aufgeteilt. Im Backend wurden die logischen Applikationskomponenten wie das Erkennen von API-Struktur und Patterns sowie das Verwalten und Persistieren von Daten mit Spring umgesetzt. Im Frontend wurde die Benutzeroberfläche und die damit verbundene Logik durch eine Angular Applikation realisiert. Diese Aufteilung der Arbeitsbereiche ermöglichte es, die Einarbeitung in die zwei Hauptbestandteile von JHipster (Springboot, Angular) ebenfalls sinnvoll aufzuteilen. Durch teaminterne Kommunikation wurde sichergestellt, dass beide Mitglieder über den aktuellen Stand beider Bereiche informiert waren, um bei Gelegenheit den Tätigkeitsbereich wechseln zu können. Die Schnittstelle zwischen Frontend und Backend wurde während der Planungsphase definiert und galt als Referenz für die Implementierung.

Das Projektteam richtete eine Entwicklungsumgebung mit Git als Versionskontrolle und automatischen Builds durch Travis ein. Die Funktionsfähigkeit der Software-Lösung wurde fortlaufend durch programmierte Tests sichergestellt. Am Ende der zweiten Implementationsphase wurde ein Usability Test mit dem Kunden durchgeführt. Die Applikation wurde im Anschluss aufgrund der Testergebnisse weiterentwickelt und verbessert. Ein zweiter Usability Test mit einem weiteren Autor der Pattern-Sprache wurde nach dieser Überarbeitung durchgeführt, um weitere Mängel auszumachen.

Während des Projektverlaufes wurden wie bereits in der Studienarbeit neue Visualisierungsentwürfe für die Pattern-Sprache entwickelt. Das Vorgehen entsprach der Studienarbeit und kann bei Bedarf der SA-Dokumentation entnommen werden.

Ergebnisse

Durch die Visualisierungsarbeit entstanden acht neue Icons. Das Feedback zu den bestehenden Icons wurde bewertet und in die Visualisierungen eingepflegt.

Das Projektteam entwickelte einen Applikationsprototyp, der API im OpenAPI Specification Format einliest und darin nach der Verwendung von Pattern sucht. In Folge diese Analyse erstellt der Dokumentationsassistent ein Modell des APIs, welchem die gefundenen Patterns zugeordnet werden. Der Benutzer kann das Resultat ergänzen und mittels Drag and Drop die Pattern-Icons, welche mit der offiziellen Website der Pattern-Sprache verknüpft sind, auf die Komponenten des APIs ziehen. So können nicht gefundene Patterns ergänzt und falsche Analyseresultate gelöscht werden. Nicht nur die Pattern-Zuordnung, sondern auch die Struktur des API ist editierbar. Auf jeder Stufe der Struktur hat der Anwender zudem die Möglichkeit, Links zu der API-Referenz zu hinterlegen. Aufgrund dieser Links und einem Keyword-Matching-Verfahren, wird in der Referenz wiederum nach Patterns gesucht, welche anschliessend angezeigt werden. Das Editieren der API Struktur macht es möglich, dass ein API auch von Grund auf aufgebaut werden kann. Somit ist die Applikation unabhängig von der OpenAPI Specification, wodurch sämtliche Web API mit dem Assistenten dargestellt werden können.

Bei Bedarf kann der Anwender die Pattern-Sammlung über die Benutzeroberfläche erweitern. So können neue Patterns hinzugefügt oder auch Keywords für das Matching editiert werden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Dieses Kapitel gibt einen Überblick über den Umfang und die Ziele des Projektes. Die unterschriebene Aufgabenstellung, welche dem Projekt zugrunde liegt, befindet sich im Anhang des Dokuments. Diese Bachelorarbeit wurde von Sebnem Kaslack und Nicolas Dipner ausgearbeitet (nachfolgend „Projektteam“ genannt).

1.1 Vorarbeit

Dieses Projekt basiert auf der Studienarbeit „Visualisierung und Umsetzung von Web API Design Patterns“ [12], welche vom Projektteam durchgeführt wurde. Schwerpunkt der Studienarbeit war das Entwerfen von Visualisierungen für die plattformübergreifende Patternsprache „Interface Representation Patterns“ (IRP). Ein Autoren-Team unter der Leitung von Herrn Zimmermann arbeitet derzeit an dieser Pattern-Sprache zur Gestaltung von APIs im Bereich der Webentwicklung. Nebst den Visualisierungen hat das Projektteam zur Unterstützung der Pattern-Autoren Known Uses der Patterns in öffentlichen Web APIs analysiert, verifiziert und dokumentiert.

1.2 Projektübersicht

Die Ergebnisse dieses Projektes wurden im Rahmen einer Bachelorarbeit an der Hochschule für Technik in Rapperswil vom Projektteam erarbeitet. Während der Bacheloarbeit wurde eine Web-Applikation entwickelt, mit welcher ein bestehendes Web API anhand der Patterns der Patternsprache für Web API Design und Evolution grafisch dargestellt werden kann. Die Applikation sucht in der Schnittstellenbeschreibung nach den Patterns und fügt Gefundene der Komponentenvisualisierung hinzu. Das Resultat der Analyse kann vom Anwender überarbeitet werden, damit nicht erkannte Patterns ergänzt und False Positives korrigiert werden können. Für die Einbindung der erstellten Visualisierung in die API-Dokumentation steht eine Exportfunktion zur Verfügung.

Eine weitere Aufgabe des Projektteams während der Bachelorarbeit war die Überarbeitung und Vervollständigung der Visualisierungen aus dem vorherigen Projekt. So wurden die bestehenden Grafiken anhand von Rückmeldungen der Pattern-Autoren überarbeitet und Entwürfe für neu entstandene Patterns gezeichnet. Eine Übersicht über alle Pattern-Icons befindet sich in Kapitel 3.

Die Entwicklung der Applikation stellt ein klassisches Software Engineering Projekt mit den Phasen Anforderungsanalyse, Design, Implementierung und Testing dar[8]. Diese Phasen werden im Einzelnen in den folgenden Kapiteln näher beschrieben.

1.3 Zielgruppe

Diese Arbeit richtet sich an API-Designer, Softwareentwickler und -architekten, die (i) sich über die vorhandenen Web API Patterns und deren Visualisierungen informieren möchten, (ii) ein bestehendes API im OpenAPI Specification-Format auf das Vorkommen von Patterns mittels einem entsprechenden Tool prüfen wollen oder (iii) ein Übersichtsdiagramm in visueller Form eines API angezeigt bekommen wollen[1].

1.4 Inhalt

Die nachfolgenden Kapitel beschäftigen sich mit den Visualisierungen von Patterns sowie dem Aufbau und Implementierung des Dokumentationsassistenten.
Die verschiedenen Themengebiete sind in Kapitel gegliedert.

Kapitel 2 beschreibt die Grundlagen von Web API Spezifikationen.

Kapitel 3 zeigt einen Überblick über die Patterns in der Patternsprache für Web API Design und Evolution

Kapitel 4 listet die funktionalen sowie nicht-funktionalen Anforderungen an die Visualisierungen und den Prototypen auf.

Kapitel 5 beschäftigt sich mit der Domainanalyse und erklärt die Bedeutung der verschiedenen Elemente innerhalb des Domain Models.

Kapitel 6 geht auf die Themen Konzeption und Design ein, stellt die Software-Architektur vor und schliesst mit Architekturentscheidungen ab.

[1] Entnommen aus dem EuroPLoP-Paper[13]

Kapitel 7 gibt Auskunft über die Schnittstelle zwischen Front- und Backend und nähere Details zu den verschiedenen Komponenten im Frontend.

Kapitel 8 zeigt auf, wie anhand eines eingelesenen OAS-Dokumentes die Pattern-Suche funktioniert, welche Patterns der Pattern-Sprache durch die Suche abgedeckt werden können und nimmt Bezug auf die Vorgehensweise bei Anpassungen von Pattern-Namen.

Kapitel 9 zeigt auf, wie anhand von Usability Tests, die Benutzeroberfläche konstant angepasst worden ist und enhält Informationen zu weiteren Testarten.

Kapitel 10 nimmt zuerst Stellung zum Endprodukt und zeigt auf, welche Anforderungen umgesetzt worden sind und schliesst mit dem Proof of Concept ab.

Kapitel 11 gibt einen Überblick und Hintergrundinformationen für Folgearbeiten.

Kapitel 12 beinhaltet einen Rückblick über die Bachelorarbeit

Kapitel 2

Technische Grundlagen

2.1 Swagger-Spezifikation 2.0

Die Swagger-Spezifikation 2.0 [2] bietet einem API-Anbieter die Möglichkeit, ein API REST-konform zu beschreiben. Wird diese Spezifikation als Grundlage für die API-Beschreibung genutzt, so müssen die API-Details, Ressourcen, die Operationen, welche auf den Ressourcen ausgeführt werden dürfen und die Nutzdaten gemäss dieser Spezifikation vom REST-API Entwickler erfasst werden. Der Vorteil ein REST-APIs auf diese Weise zu beschreiben ist, dass sie von Menschen gut verstanden werden können, ohne einen Einblick in den Quellcode zu haben. Swagger erleichtert einem API-Entwickler durch zusätzliche Tools wie Swagger Editor oder Swagger UI die Erstellung eines APIs.

2.2 OpenAPI Spezifikation 3.0.0

Die Swagger-Spezifikation wurde von Smartbear[1] am 1. Januar 2016 der Open API Initiative (OAI) [2] übergeben. Diese Initiative hat zum Ziel die anbieterneutrale Standardisierung von REST APIs voranzutreiben. Aufgrund dieser Übergabe wurde der Name geändert (von Swagger 2.0 zu OAS 3.0.0). OAS 3.0.0 ist eine Weiterentwicklung von Swagger 2.0 und ist mit neuen Funktionalitäten ausgerüstet, weshalb die Laufnummer der Version weitergeführt wurde. Die neue Version mit dem neuen Funktionsumfang wurde von der Open API Initiative am 27. Juli 2017 vorgestellt [1].

[1] https://smartbear.com/
[2] https://www.openapis.org/

Kapitel 3

Patternsprache für Web API Design und Evolution

3.1 Einleitung

Die Implementierung einer Web-Applikation steht im Vordergrund der vorliegenden Bachelorarbeit. Ein Ziel eines Anwenders dieser Applikation ist es, ein API einzulesen und zu analysieren und als Resultat eine grafische Übersicht über die in der API gefundenen Patterns zu erhalten. Diese Patterns bilden die Grundlage für die Bachelorarbeit und sind zentraler Bestandteil der Web-Applikation. Zurzeit arbeitet das Autorenteam des WADE-Projektes an einer Patternsprache, welche die eben genannten Patterns umfasst. Die Patterns werden fortlaufend überarbeitet und durch neue Entwürfe ergänzt.

3.2 Pattern-Kategorien

Jedes Pattern ist einer Kategorie aus der Patternsprache zugeordnet. Tabelle 3.2.1 zeigt die Kategorie-Icons auf.

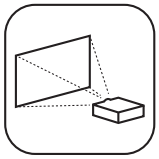
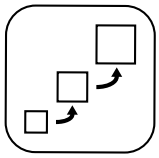
Pattern Kategorie Icons				
 Foundation	 Structural Representation	 Pagination	 Responsibility	 Quality
 Evolution				

Tabelle 3.2.1: „Übersicht der Kategorien-Icons“

3.3 Pattern-Übersicht

Nachfolgend werden die Patterns in tabellarischer Form pro Pattern-Kategorie aufgelistet. Die während der Bachelor-Arbeit vom Projektteam neu ausgearbeiteten Icons sind mit „neu“ gekennzeichnet. Die Pattern-Icons und Namensgebungen der Patterns können von denen, welche in der Studienarbeit von S. Kaslack und N. Dipner [12] festgehalten wurden, abweichen. Diese Korrekturen entstanden anfangs Projektstart aufgrund der Rückmeldung der Pattern-Autoren und deren Anpassungs- bzw. Änderungswünschen. Im Absatz 3.4 sind die ausgeführten Anpassungen aufgelistet.

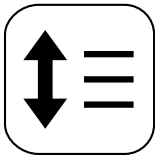
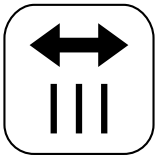
Foundation Patterns				
 Public API	 Community API	 Solution-Internal API	 Vertical Integration	 Horizontal Integration
 Service Contract				

Tabelle 3.3.1: Patterns der Kategorie „Foundation“

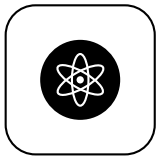
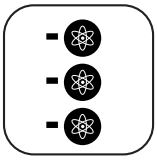
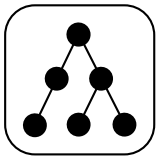
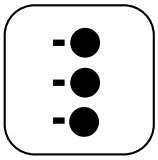
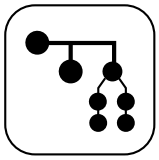
Structural Representation Patterns				
 Atomic Parameter **überarbeitet**	 Atomic Parameter List **überarbeitet**	 Parameter Tree	 Parameter List **überarbeitet**	 Parameter Forest

Tabelle 3.3.2: Patterns der Kategorie „Structural Representation“

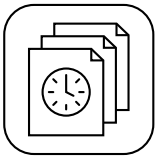
Pagination Patterns				
 Cursor-Based Pagination	 Offset-Based Pagination	 Time-Based Pagination		

Tabelle 3.3.3: Patterns der Kategorie „Pagination“

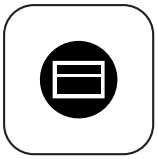
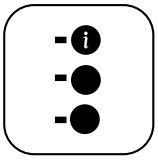
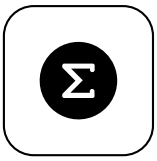
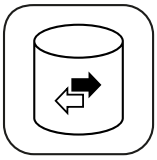
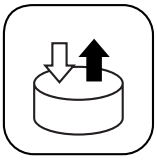
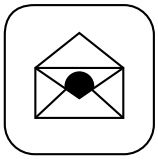
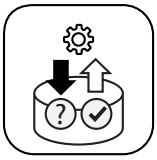
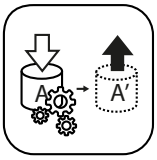
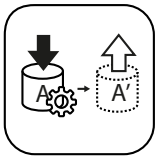
Responsibility Patterns				
Unterkategorie: Parameter Types				
 Entity Parameter	 Id Parameter	 Link Parameter	 Metadata Parameter	 Annotated Parameter Collection
Unterkategorie: Metadata Parameter				
 Provenance Metadata	 Control Metadata	 Aggregated Metadata		
Unterkategorie: Resource Types				
 Master Data Resource	 Transactional Data Resource	 Data Lookup Resource (**neu**)	 Embedded Reference Data	 Linked Reference Data
Unterkategorie: Processing Types				
 Query Service (**neu**)	 Validation Service (**neu**)	 Business Activity Service (**neu**)	 Event Notification Service (**neu**)	

Tabelle 3.3.4: Patterns der Kategorie „Responsibility“

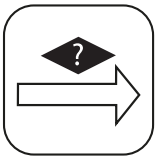
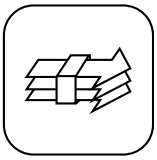
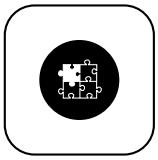
Quality Patterns				
 Service Level Agreement	 Metering and Billing (**neu**)	 API Key	 Rate Limit	 Wish List
 Wish Template	 Conditional Request (**neu**)	 Request Bundle	 Context Representation (**neu**)	 Error Reporting

Tabelle 3.3.5: Patterns der Kategorie „Quality“

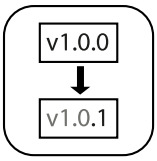
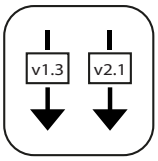
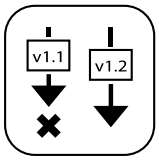
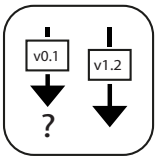
Evolution Patterns				
 Version Identifier	 Semantic Versioning	 Eternal Lifetime Guarantee	 Limited Lifetime Guarantee	 Two In Production
 Aggressive Deprecation	 Experimental Preview			

Tabelle 3.3.6: Patterns der Kategorie „Evolution“

3.4 Änderungsübersicht

Die vom Projektteam vorgenommenen Anpassungen (Name und Icon) sind nachfolgend aufgeführt:

Pattern- und Kategoriename	**Bemerkung**
Annotated Parameter Collection	Umbennenung Pattern (vorher: Annotated Parameter List)
Atomic Parameter	Anpassung Icon gemäss Change Request
Atomic Parameter List	Anpassung Icon gemäss Change Reuqest
Eternal/Limited Lifetime Guarantee	Lifetime Guarantee aufgesplittet in zwei Patterns
Master Data Resource	Umbennenung Pattern (vorher: Master Data)
Offset-Based Pagination	Umbenennung Pattern (vorher: Page-Based Pagination)
Parameter Forest	Umbennung Pattern (vorher: Parameter Comb)
Quality Patterns	Umbenennung Kategorie (vorher: QoS Patterns)
Responsibility Patterns	Umbennenung Kategorie (vorher: Semantic Patterns)
Service Level Agreement	Umbennung Pattern (vorher: SLA-SLO)
Structural Representation Patterns	Umbennung Kategorie (vorher: Basic Representation Patterns)
Transactional Data Resource	Umbennung Pattern (vorher: Transaction Data)
Two in Production	Anpassung Icon gemäss Change Request
Wish Template	Umbenennung Pattern (vorher: Wish Object) und Anpassung Icon gemäss Change Request

Tabelle 3.4.1: Änderungsübersicht Visualisierungen

Kapitel 4

Anforderungen

Dieses Kapitel beschreibt die Anforderungen sowohl an den Prototyp wie auch an die Visualisierungen. Die Anforderungen an die Visualisierungen wurden zwecks Vollständigkeit aus dem Abschnitt „Qualitätsmerkmale“ des Visualisierungskonzept der Studienarbeit[12] entnommen. Die Visualisierungsarbeiten der Bachelorarbeit folgen diesem Konzept. Die Merkmale werden in gekürzter Form aufgeführt. Auf Erläuterungen zu diesen Anforderungen wird verzichtet, da diese aus der Studienarbeit abgeleitet werden können und der Fokus dieser Arbeit auf dem Prototyp liegt.

4.1 Qualitätsmerkmale der Visualisierungen

Folgende Qualitätsmerkmale wurden während der Studienarbeit in Kooperation mit Herrn Zimmermann erfasst [12].

Visuelle Konsistenz
Die Visualisierungen haben einen klar ersichtlichen einheitlichen Stil. Die verwendeten Elemente der Visualisierungen folgen demselben Grundprinzip.

Abstraktionsgrad
Die Visualisierungen haben denselben Abstraktionsgrad, d. h. Beispiele aus anderen Themengebieten zur Verdeutlichung sind sinnvoll und auf einem vergleichbaren Level gewählt.

Unterscheidbarkeit
Unterschiede zwischen den verschiedenen Patterns müssen klar erkenntlich sein. Ähnliche Darstellungen sind nur dann zugelassen, wenn die Unterscheidung effektiv gerechtfertigt ist.

Wiedererkennungswert
Die Visualisierungen haben einen potentiellen Wiedererkennungswert, damit Betrachter die Darstellungen in Zukunft implizit mit den IRP in Verbindung setzen und die Patterns somit besser memorieren können.

Farbkonzept
Die Visualisierungen sollen auch in Schwarz/Weiss funktionieren, ohne dass der Informationsgehalt abnimmt. Farbenblinde und Personen mit anderen Seheinschränkungen sollen durch die Farbwahl nicht benachteiligt werden.

Multi-Channel Verwendung
Die Visualisierungen werden auf verschiedenen Kanälen eingesetzt und sollen entsprechend darauf ausgerichtet sein.

International verwendbar
Zielgruppe sind Leser der ganzen Welt, entsprechend sollen global verständliche Sujets verwendet werden.

Political Correctness
Die Illustrationen beleidigen in keiner Weise eine bestimmte Gruppe von Menschen.

Skalierbarkeit
Die Darstellungen sollen gut skaliert werden können, ohne in der Optik oder Aussagekraft an Qualität zu verlieren.

Zweidimensionale Darstellung
Dreidimensionale Darstellungen sind für die Patternsprache ungeeignet und bringen keinen Mehrwert.

Erweiterbarkeit
Die Patternsprache ist nicht abschliessend, die Visualisierungen sollen eine einfache Erweiterung unterstützen. Neue Patterns lassen sich leicht integrieren.

Pattern-Kategorien
Aus der Visualisierung eines Patterns soll die Zuordnung zu der jeweiligen Pattern-Kategorie ersichtlich sein.

Lizenzen
Es sollen lediglich geeignete OpenSource-Lizenzen verwendet werden (wenn möglich Creative Commons, Apache 2 oder Eclipse), um den Einsatz der Sprache zu erleichtern. Die Verwendung von BSD und MIT wird je nach Umstand geduldet.

4.2 Anforderungen an den Prototyp

Dieser Abschnitt beschreibt Anforderungen, die der Prototyp erfüllen sollte.

4.2.1 Funktionale Anforderungen

Die Szenarien in den User Stories legen den Funktionsumfang des Prototyps fest. Das Endprodukt soll sämtliche hier festgelegte User Stories abdecken [14].

Story 1: API Dokumentation anhand der Icons

Als **API Designer** möchte ich hierarchisch aufeinander bauende Diagramme mit Hilfe der Pattern-Icons erstellen. Diese grafischen Übersichten sollen mich in der Konzeption eines APIs unterstützen.

Die Applikation sollte so flexibel sein, dass eine API Struktur von Grund auf aufgebaut werden kann. Dieser Vorgang sollte unabhängig von der im API verwendeten Plattform bzw. Technologie möglich sein.

Story 2: Analyse eines bestehenden APIs im OAS-Format

Als **Applikationsentwickler** möchte ich ein bestehendes API (im OpenAPI-Specification Standard) mit dem Visualisierungstool analysieren lassen, um mir möglichst viele der darin gefundenen Patterns anzeigen zu lassen.

Die Struktur eines APIs in einem klar definierten Format wie OAS, sollte automatisiert dargestellt werden können. Anhand des Aufbaus können bereits Schlüsse für das Pattern-Finding gezogen werden. An die Analyseresultate von solch einer maschinenlesbaren Beschreibung ist die Erwartungshaltung entsprechend grösser als an Freitextbeschreibungen.

Story 3: Analyse eines bestehenden APIs mit Freitextbeschreibung

Als **API Designer** möchte ich ein bestehendes API (nicht basierend auf ein Standard) mit dem Visualisierungstool analysieren lassen, um einen ersten Eindruck zu erhalten, ob die Patterns im API gefunden werden.

Da zahlreiche Web API-Beschreibungen keiner Spezifikation folgen, muss die Applikation Freitextbeschreibungen unterstützen, auch wenn dies bedeutet, dass in solchen Fällen mehr Benutzerinteraktion benötigt wird.

Story 4: Editieren des Analyseergebnisses

Als **API Designer** und Anwender des Visualisierungstools, will ich den Output bzw. das Ergebnis des Pattern Finding kommentieren und ergänzen können.

False-Positives oder nicht gefundene Patterns, welche durch den Benutzer erkannt werden, müssen korrigiert werden können. Des weiteren soll der Benutzer die Grafik möglichst flexibel ausbauen können.

Story 5: Pattern-Informationen

Als **Applikationsentwickler** möchte ich im Output des Pattern Finding mehr Informationen zu den einzelnen Patterns angezeigt bekommen. Einerseits möchte ich sehen, wo im API das Pattern vorkommt und andererseits eine Referenz auf die Patternschreibung, um die Funktionsweise eines API besser verstehen zu können.

Die Applikation soll dem Anwender den Einstieg in die Patternsprache erleichtern, indem Informationen zu den Patterns direkt am entsprechenden Ort bezogen werden können. Durch das Erkennen der Pattern im eigenen API wird das Verständnis der Sprache gefördert.

Story 6: Navigation durch die API Struktur

Als **API Designer und Reviewer** will ich eine detaillierte visuelle Übersicht des eingelesenen APIs, in der ich über eine Grobstruktur in die innere Struktur des APIs hineinzoomen kann, um eine Überladung der Icons auf einer visuellen Ebene zu vermeiden.

Eine übersichtliche Darstellung ist unentbehrlich für eine gute User Experience. Der Anwender soll sich auf das wesentliche Konzentrieren können und entsprechen schnell durch die Applikation navigieren können.

Story 7: Schnelle Erfassung grundlegender Eigenschaften

Als **übergeordneter IT-Architekt**, der verschiedene APIs untersucht und wesentliche Teile eines APIs schnell verstehen will, möchte ich die wichtigsten Eigenschaften eines API aus den verschiedenen visuellen Darstellungen schnell verstehen können, um diese Art der API Beschreibung für zukünftige Arbeiten einzusetzen und als Diskussionsgrundlage zu verwenden.

Die Stärke von Grafiken ist die Informationsübermittlung. Durch die Darstellung eines APIs mit den Pattersn ist es möglich, ohne viel Worte die wesentlichen Funktionalitäten zu Ausdruck zu bringen. Die Patternsprache wie auch der Prototyp soll davon gekennzeichnet sein.

Story 8: Vorschläge für die Verwendung weiterer Patterns

Als **API Designer** möchte ich ein bestehendes API mit dem Tool analysieren können und Vorschläge erhalten, mit welchen Pattern ich das API und dessen Handhabung optimieren kann.

Durch die Umsetzung der Patterns im API wird eine Applikation robuster. Der Dokumentationsassist soll daher wenn möglich Vorschläge für die Umsetzung von nicht verwendeten Pattern anzeigen.

Story 9: Einbinden der Visualisierungen in die API-Dokumentation

Als **API Designer** möchte ich die durch das Tool erstellten Darstellungen in der Dokumentation meines API an der entsprechenden Stelle einbinden, damit sie den API-Nutzern zu Verfügung stehen.

Abbildung 4.2.1 zeigt eine Übersicht der zuvor beschriebenen User Stories.

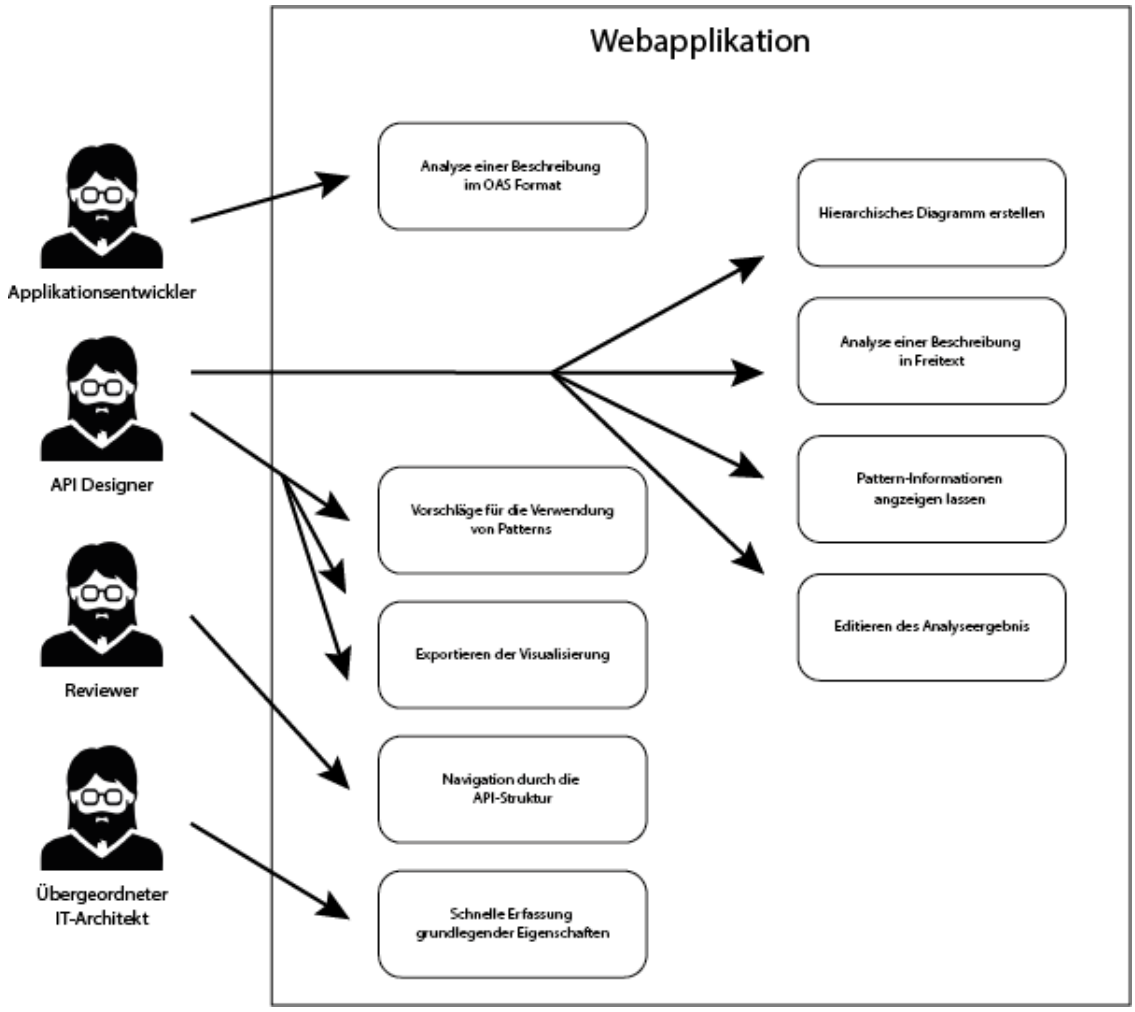


Abbildung 4.2.1: User Story Diagramm

4.2.2 Nicht-funktionale Anforderungen

Für die nachfolgende Analyse werden mehrheitlich die Qualitätsattribute gemäss ISO 9126 bzw. ISO 25010 in Betracht gezogen.

Funktionalität

Güte der Analyseergebnisse
Das Pattern Finding soll präzise und keine irrelevanten Ergebnisse liefern. Im Fokus steht die Genauigkeit der Analyseergebnisse.

Richtigkeit
Bei der Visualisierung eines APIs ist sicherzustellen, dass die Pattern-Icons, welche bei der Analyse gemappt wurden, korrekt und übersichtlich dargestellt werden.

Benutzbarkeit

Verständlichkeit
Das Visualisierungstool soll ohne Erklärungstext (im User Interface) auskommen. Der Anwender soll die drei Funktionsebenen (Pattern Finding, Visual Representation und Recommendation) auf Anhieb wahrnehmen. Wo nötig, werden Erklärungstexte angezeigt. Kommen Fachausdrücke vor, werden diese weiter erläutert.

Erlernbarkeit und Bedienbarkeit
Der Aufwand für einen Anwender das Tool zu erlernen und zu verstehen soll nicht mehr als 30 Minuten betragen.

Feedback
Bei der Ausführung einer Aktion, ist klar, welches Ergebnis geliefert werden soll. Im Falle eines Fehlers wird der Benutzer entsprechend informiert. Tritt ein Fehler ausserhalb des Use Cases auf, wird dies durch einen allgemeinen Error-Handler in der Anwendung abgefangen.

Zuverlässigkeit

Robustheit
Fehlerhafte Benutzer- oder Dateneingaben werden abgefangen und sollen zu keinen Systemabstürzen führen. Tritt ein nicht abgefangener Fehler auf, soll dies nicht zu Datenverlusten führen.

Effizienz

Zeitverhalten
Die Antwortzeit bei der Analyse eines Patterns soll in 80% der Anwendungsfälle nicht länger als 8 Sekunden dauern.

Verbrauchsverhalten
Das Visualisierungstool soll auf modernen Windows-Betriebssystemen (ab Windows 7) und Linux-Betriebssystemen lauffähig sein.

Die Web-Applikation soll im Firefox Webbrowser (ab Version 55) laufen.

Übertragbarkeit

Der Benutzer hat minimalen Konfigurationsaufwand beim Deployen der Applikation unter der Annahme, dass das Deployment an einem HSR Standard Entwickler-PC vollzogen wird.
Mit der Installationsanleitung kann das Deployment in 10 Minuten erfolgen.

4.2.3 Randbedingungen

Der Einsatz von Frameworks ist erlaubt, solange die darunterliegenden Lizenzen den nicht-funktionalen Anforderungen entsprechen.

Für die Entwicklung des Visualisierungstools wurde vom Projektbetreuer das Java Framework JHipster empfohlen, eine Entwicklungsplatform, um Web-Applikationen basierend auf Spring Boot und AngularJS zu erstellen.

Lizenzen

Es sollen lediglich geeignete OpenSource-Lizenzen verwendet werden (wenn möglich Creative Commons, Apache 2 oder Eclipse), um die Verwendung der Sprache zu erleichtern. Die Verwendung von BSD und MIT wird je nach Umstand geduldet. Zu vermeiden sind die Lizenzmodelle GPL und LGPL, da diese oftmals ein no-go für Firmen sind. Die Visualisierungen sollen allgemein möglich Lizenz-unabhängig sein.

Kapitel 5

Domainanalyse

Dieses Kapitel befasst sich mit der Problemdomäne. Für die Applikation relevante Objekte werden definiert und analysiert. Abbildung 5.0.1 stellt den abstrakten Aufbau des Lösungsvorschlags für die Applikation dar und umfasst die wichtigsten Objekte der Domäne. Die Objekte lassen sich anhand ihrer Herkunft in drei Bereiche unterteilen:

1. **API**: Daten aus der API-Beschreibung
2. **Visualisierung**: Output-Daten der zu entwickelnden Applikation
3. **Patternsprache**: Daten aus den Patternbeschreibungen und Visualisierungsarbeit

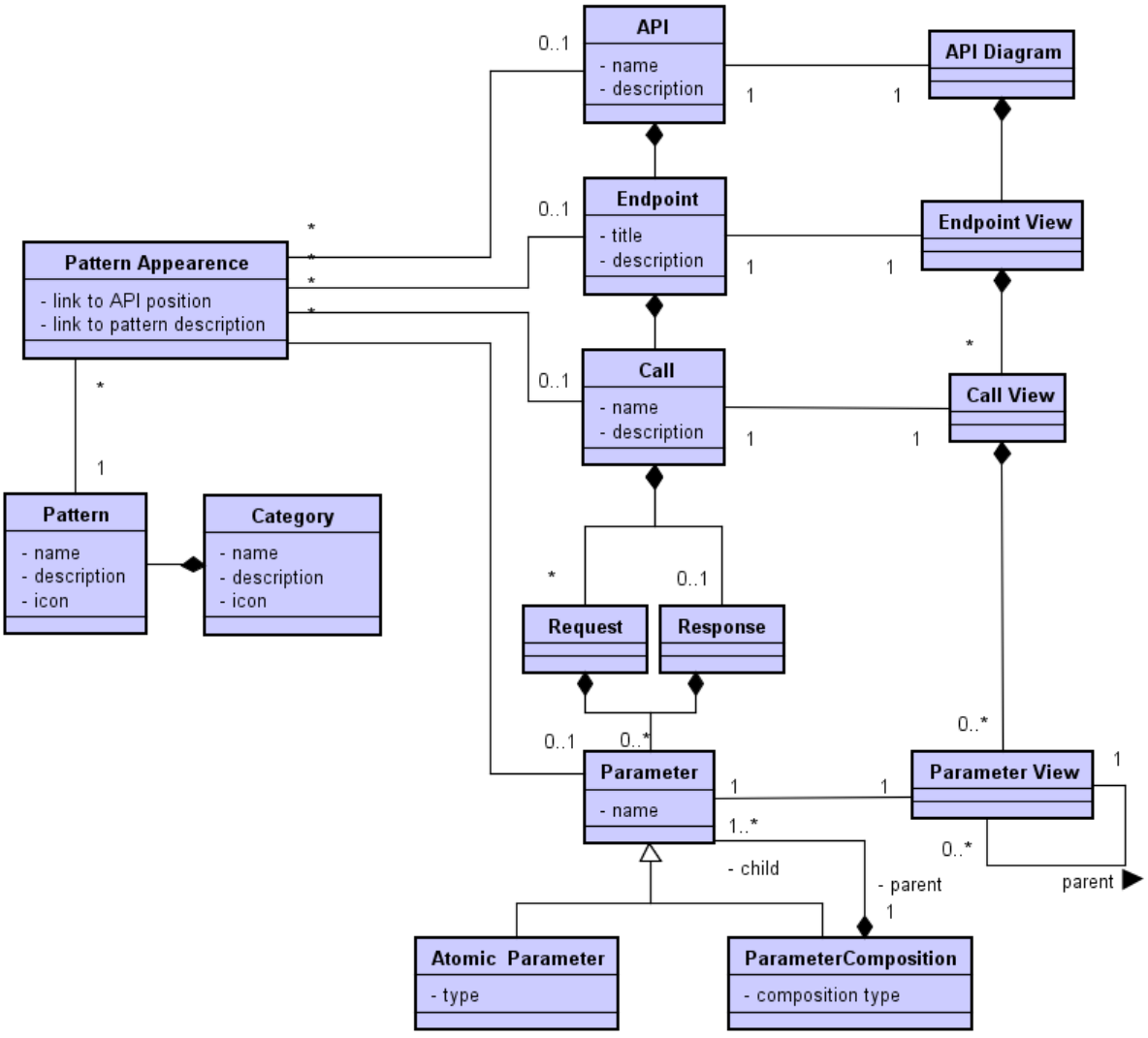


Abbildung 5.0.1: Domain Model

5.1 API-Struktur

Die Applikation erstellt eine API Struktur anhand einer Quelldatei. Die Quelle ist im Rahmen der Domänenanalyse als eine Schnittstellenbeschreibung mit unbestimmten Format zu behandeln. Informationen aus der Quelle werden in der Applikation auf eine vorgegebene API Struktur gemappt. Die API Struktur kann aus dem Domänenmodel entnommen werden, sie umfasst Komponenten auf vier Ebenen (API, Endpoint, Call und Parameter).

Die vier Begriffe API, Endpoint, Call und Parameter werden in der Patternsprache verwendet und beschreiben die für die Webapplikation definierte Hierachie eines APIs. Diese Hierarchie entstand durch die Analyse von API-Beschreibungen im OAS Format und wurde anhand der WSDL- und der REST-Definitionen auf ihre Verwendbarkeit in Bezug auf das plattformunabhängige Mapping von Informationen geprüft.

Tabelle 5.1.1 zeigt vergleichbare Hierarchiestufen in WSDL und REST. Mit diesem Verständnis lässt sich mit der Applikation plattformunabhängig eine API-Struktur abbilden, auch wenn durch das generische Design der Lösung einzelne Eigenheiten spezifischer Implementationen abgebildet werden können.

Ebene	**Patternsprache**	**WSDL**	**REST**
1	API	API	API
2	Endpoint	Endpoint (ab WSDL 2.0 vorher PortType)	(Home-)Resource
3	Call	Operation	HTTP Method (Verb)
4	Parameter	Messages (input/output)	Representation (Objetct, Attribute, Media Type, Schema)

Tabelle 5.1.1: Gegenüberstellung Patternsprache, WSDL und REST

API: Die API-Ebene ist das Wurzelelement einer der API-Struktur. Sie beinhaltet allgemeine Informationen zum API und ist der Einstiegspunkt der API-Referenz.

Endpoint: Auf dieser Ebene verläuft der Zugriff auf eine bestimmte Entität oder einen bestimmten Service. Ein Endpoint verfügt unter Umständen über mehrere Möglichkeiten, Daten abzurufen. Dies geschieht über Calls.

Call: Der Call bezeichnet die effektive Funktion eines API-Aufrufes. Im Domainmo-

del ist dem Call jeweils ein Request und ein Reply zugeordnet. Diese Unterteilung ermöglicht die Separierung der Parameter, die sich auf Aufruf und Antwort eines Calls verteilen.

Parameter: Die Parameter sind Input und Output eines Calls und beinhalten sowohl Objekte, Attribute als auch das Schema einer Message und können daher beliebig geschachtelt sein. Daher wurde an dieser Stelle im Domänenmodell das Composite-Pattern angewandt.

5.2 Pattern und Pattern Appearance

Die Patterns werden durch das Autorenteam der Patternsprache ausgearbeitet. In der Applikation werden diese Patterns durch die Visualisierungen aus der vorgängigen Studienarbeit repräsentiert und mit der Patternbeschreibung der offiziellen Website verknüpft. Aufgabe der Applikation ist es, die API-Struktur durch Patterns zu ergänzen. Hierfür wurde die Pattern Appearance definiert. Eine Pattern Appearance ist die Zuordnung eines konkreten Patterns mit einer konkreten API-Komponente. Somit lösen die Appearances die m-zu-n-Beziehungen zwischen Komponenten und Patterns auf und bilden zusammengefasst eine Gesamtübersicht über die Patterns, welche im API gefunden bzw. zugeordnet wurden.

5.3 API Diagramm

Endresultat der Applikation ist eine Visualisierung des APIs, ein Diagramm, welches die wesentlichen Eigenschaften des API darstellt. Die Anforderungen an die Applikation erfordern, dass der Output der Analyse editierbar sein soll, durch die Komponenten navigiert und ein API flexibel aufgebaut und erweitert werden kann. Entsprechend wird für jede Komponente eine View erfasst, welche an die Komponente gebunden ist.

Kapitel 6

Konzeption und Design

6.1 Software-Architektur

Für die Beschreibung und Kommunikation der Architektur wird nachfolgend das C4-Architekturmodell verwendet. Durch das Modell wird das Design der Software auf einfache Weise allen involvierten Parteien in Form von Diagrammen bzw. Grafiken verständlich gemacht. Die Diagramme sollen so viele Informationen enthalten, dass der Betrachter schnell eine Übersicht des Systems bzw. der Architektur enthält.

Das C4 Modell beinhaltet vier Typen von Diagrammen:

- context diagram
- container diagram
- component diagram
- class diagram(s)

6.1.1 Kontextdiagramm

Für das Erstellen des Kontextdiagramms und dessen Abgrenzung wurden die Resultate der Anforderungsanalyse berücksichtigt. Gernot Starke, Autor des Buches „Effektive Softwarearchitekturen“[11], behandelt im Kapitel 5.5 die Kontextabgrenzung auf gleiche Weise wie das C4 Kontext Abgrenzung. Er führt aus:

> „Die Kontextabgrenzung zeigt das Umfeld eines Systems sowie dessen Zusammenhang mit seiner Umwelt. In diesem Sinne ist die Kontextabgrenzung eine Vogelperspektive oder gesamthafte Übersicht.[1] Sie zeigt

[1] In der Literatur finden Sie auch folgende Bezeichnungen: „Kontextsicht“ [iSAQB] oder „Conceptual Architecture View“ [Hofmeister2000].

das System als Blackbox sowie dessen Verbindungen und Schnittstellen zur Umwelt. Es ist eine Sicht auf hoher Abstraktionsebene“.

Die Abbildung 6.1.1 zeigt den Kontextdiagramm für die Web-Applikation.

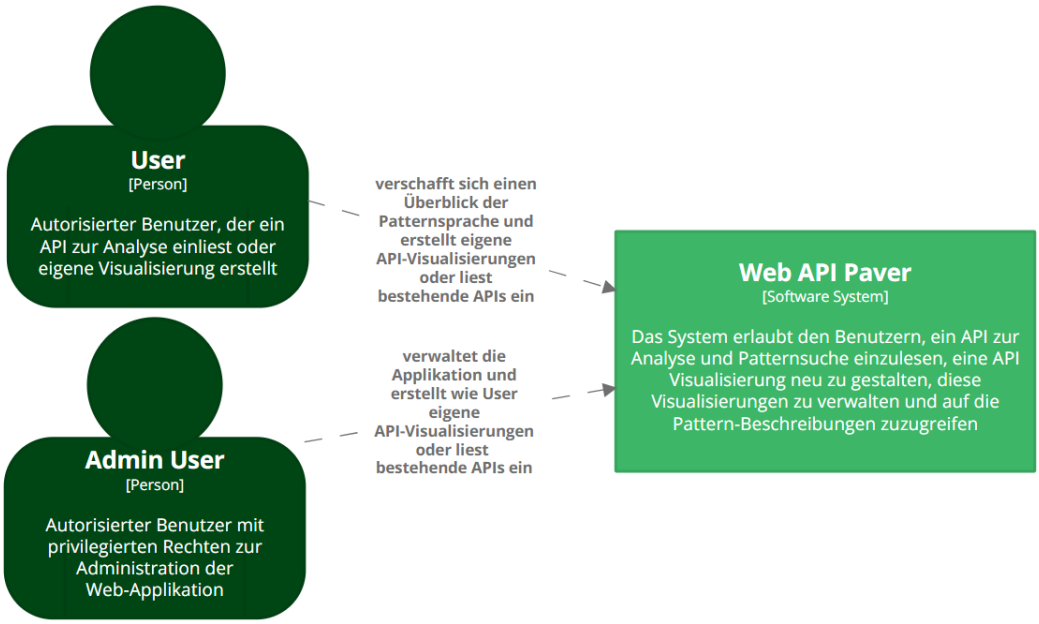


Abbildung 6.1.1: Context Diagram

Es gibt zwei Typen von Anwendern, die mit dem System interagieren. Die Applikation selbst kommuniziert nicht mit anderen internen oder externen Systemen.

6.1.2 Containerdiagramm

Mit dem Containerdiagramm zoomt man in das System hinein. Ein Container kann gemäss dem C4 Modell eine Web-Applikation, Desktop-Applikation, Mobile-Applikation, Datenbank, usw. sein[3]. Mit diesem Diagrammtyp kann eine Aussage darüber gemacht werden, wie das Software-System aussehen soll, welche High-level Technologie-Entscheidungen getroffen wurden und wie verschiedene Container miteinander kommunizieren.[2]

Die Abbildung 6.1.2 zeigt das Containerdiagramm für die Web-Applikation.

[2] https://c4model.com/

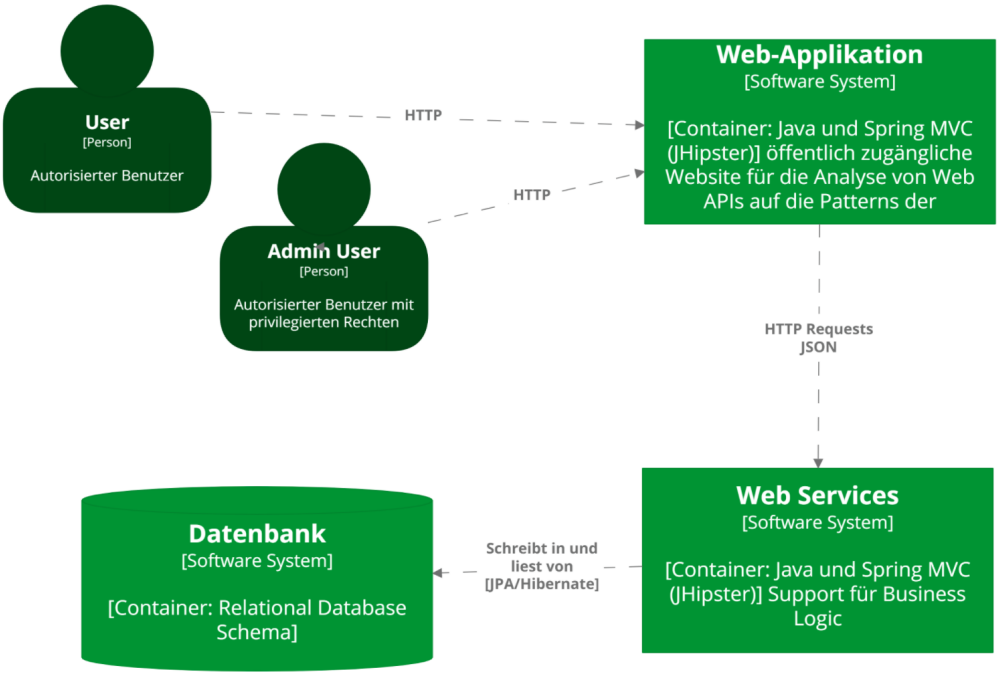


Abbildung 6.1.2: Container Diagram

Für das Erstellen einer Basis-Applikation kommt JHipster[4] zum Einsatz. JHipster generiert eine komplette und auf den neuesten Technologien basierende Web-Applikation. Bootstrap und AngularJS im Frontend und Spring Boot im Backend werden durch JHipster zusammengebracht. Auf Basis dieser Applikation wurde der Dokumentationsassistent implementiert und beinhaltet somit standardmässig Spring MVC. Durch den Einsatz von Spring Boot ist das Aufsetzen eines Applikationsservers nicht notwendig, um die Web-Applikation zu deployen, was den Initialaufwand wiederum verringert.

6.1.3 Komponentendiagramm

Das Komponentendiagramm zeigt das Innere eines Containers auf. Ein Container kann mehrere Komponenten beinhalten. Komponenten können nach deren Funktionalität gruppiert werden. Die offizielle Beschreibung auf `https://c4model.com` lautet:

> „If you’re using a language like Java or C#, the simplest way to think of a component is that it’s a collection of implementation classes behind an interface.“.

Dieser Diagrammtyp eignet sich für technisch versierte Personen innerhalb eines Entwicklungsteams. Die Abbildung 6.1.3 zeigt die Komponenten der Web-Applikation, die für das Erstellen einer API-Beschreibung notwendig sind. Der Anwender

kann mit diesen Komponenten eine API-Visualisierung erstellen. Für die Pattern-Zuweisung werden die in Abbildung 6.1.4 aufgelisteten Entitäten verwendet. Diese greifen wie in derAbbildung 6.1.3 gezeigt, über den Service Layer auf die Repositories zu.

Die Abbildung ist eine grobe Übersicht und zeigt auf, dass die Web-Applikation in drei Schichten (Repository und Data Access als eine Ebene betrachtet) organisiert ist (Presentation Layer, Service Layer, Repository Layer, Data Access). Die Controller entsprechen der im Domainmodell festgehaltenen Struktur. Für die Erstellung einer eigenen API-Visualisierung oder Hinzufügen von weiteren Ebenen auf eine bestehendes API, werden die in Abbildung 6.1.3 und in Abbildung 6.1.4 aufgelisteten Controller angesprochen, hauptsächlich für CRUD-Operationen.

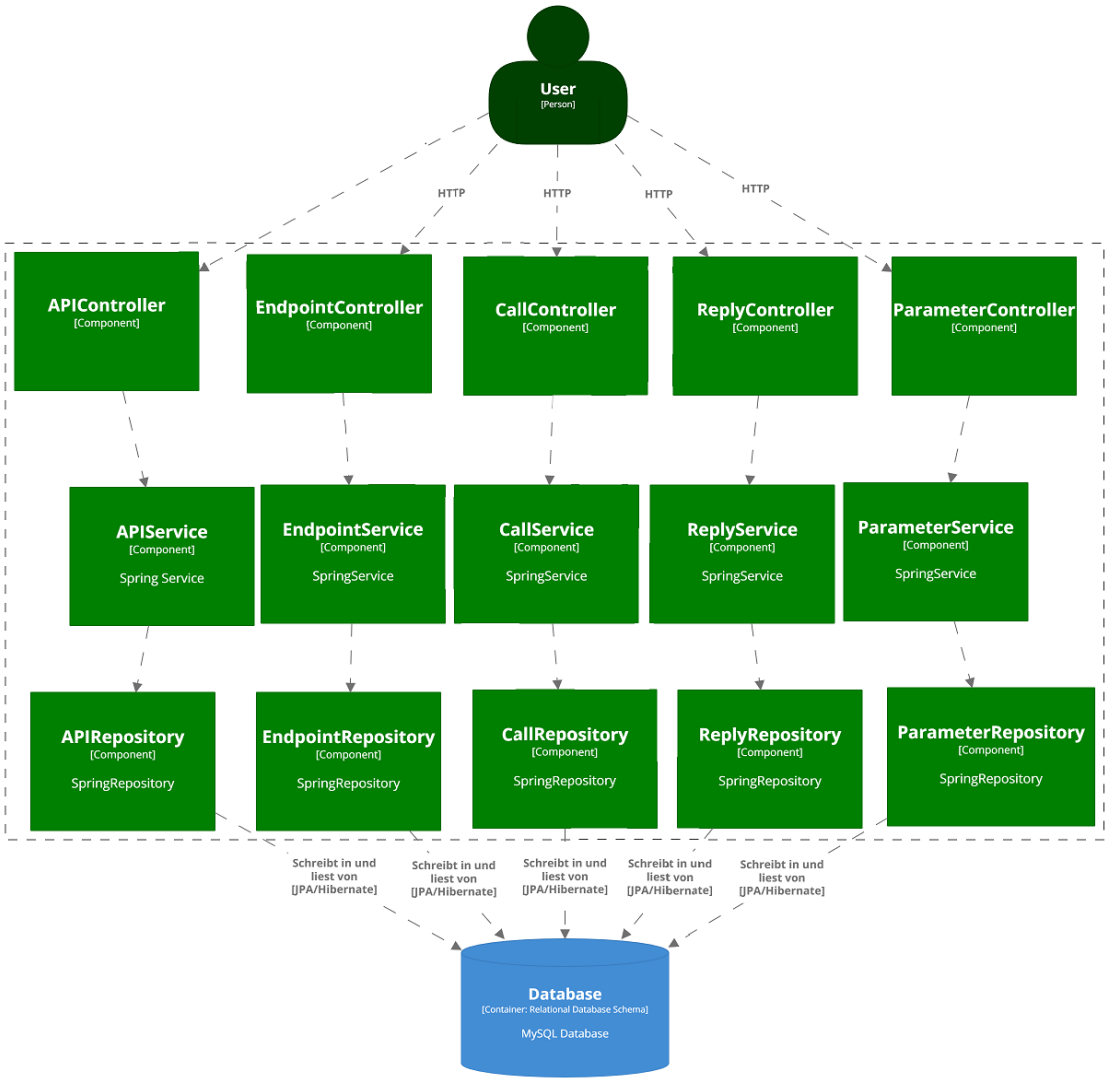


Abbildung 6.1.3: Container Diagram

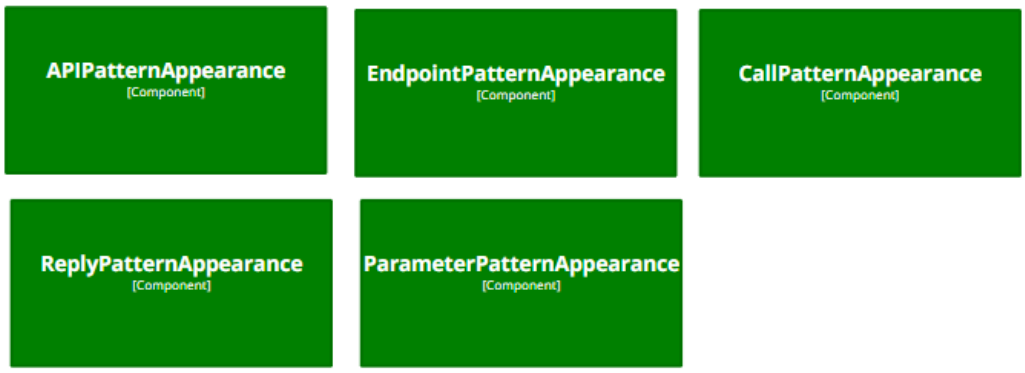


Abbildung 6.1.4: Container Diagram

Wird ein existierendes API im richtigen Format zur Analyse eingelesen, so wird der `UploadRequestController` angesprochen. Das Parsen des Swagger-Dokuments geschieht über die entsprechenden Service Parser-Klassen, die Zuordnung der Patterns erfolgt über die Service Pattern-Appearance-Klassen.

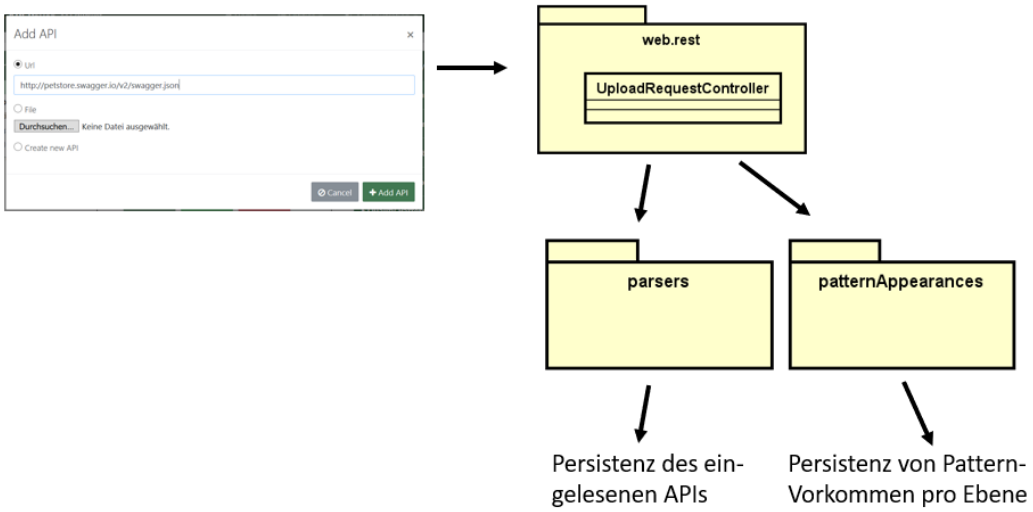


Abbildung 6.1.5: Übersicht Upload Swagger API Beschreibung

6.1.4 Architekturentscheidungen

Der Inhalt des Kapitels 4.2 Anforderungen an den Prototyp fliesst in die Architekturentscheidungen mit ein. Relevante Punkte sind unter anderem, dass der Dokumentationsassistent (i) eine grafische Oberfläche hat, über welche ein API zur Analyse eingelesen oder eine neue API-Visualisierung erstellt werden kann, (ii) keine Fremdsysteme mit einbindet, (iii) keine Anforderungen zur Persistenz bestehen.

Entscheidung 1: Framework

Wie im Absatz 4.2.3 erläutert, kam die JHipster-Umgebung als möglicher Kandidat in Frage, auf welcher der Dokumentationsassistent aufgebaut wird. Die Tabelle 6.1.1 gibt Aufschluss über die Framework-Entscheidung.

Subjektdomäne	Software Architecture Design
Name	Framework-Auswahl
Getroffene Entscheidungen	Das Projektteam hat sich nach einer kurzer Evaluationsphase für JHipster entschieden. JHipster basiert auf Spring Boot im Backend und AngularJS im Frontend.
Problem	Es soll entschieden werden, welches Java-Framework für die Implementierung der Applikation in Frage kommt.
Annahme	Das System wird nicht von Grund auf neu implementiert. Durch den Einsatz eines Frameworks hat man bei der Implementierung den Vorteil, dass der Initialaufwand geringer ist.
Motivation	Durch die Auswahl des richtigen Frameworks kann sich das Projekteam auf das Wesentliche konzentrieren und muss sich zum Beispiel nicht selbst um OR-Mapping kümmern.
Alternativen	Evaluiert wurden das Spring und Play Framework.
Begründung	JHipster bietet die Möglichkeit, eine Basis Web-Applikation zu generieren, welche Front- und Backend vereint. Dem Projektteam war es wichtig, mit einer Basisapplikation zu arbeiten und diese von Hand weiterzuentwickeln.
Folgen	Da das System neu erstellt wird, zieht das keine Konsequenzen auf bestehende Architekturbestandteile. Bei der Implementierung ist darauf zu achten, die Richtlinien von JHipster zu beachten.

Tabelle 6.1.1: Architekturentscheid für JHipster

Entscheidung 2: Persistenz

Die Tabelle 6.1.2 gibt Auskunft über den Architekturentscheid zur Persistenz:

Subjektdomäne	Persistenz
Name	Datenbank-Auswahl
Getroffene Entscheidungen	Das Projektteam hat sich für die Datenbanklösung MySQL entschieden.
Problem	JHipster schränkt den Benutzer bei der der Wahl der Datenbank nicht ein, jedoch sind gewisse Datenbanken standardmässig nicht vorkonfiguriert und müssen selbst angebunden werden. Der Dokumentationsassistent braucht ein Persistierung-Layer. Anforderungen, welche für die Applikation relevant sind, bestehen nicht.
Annahmen	In den Anforderungen besteht keine Angabe zum gleichzeitigen Zugriff auf die Applikation von mehreren Anwendern oder zu Verfügbarkeit.
Motivation	Für die Web-Applikation und deren Funktionalität spielt es im Data Layer keine Rolle, welcher Datenbanktyp ausgewählt wird. Da mit JPA und Hibernate Entitäten und deren Relationen in Tabellen abgebildet werden, ist es naheliegend, eine relationale Datenbank einzusetzen.
Alternativen	PostgreSQL
Begründung	Das Projektteam hat mehr Erfahrung mit MySQL und hat somit einen geringeren Initialaufwand.

Tabelle 6.1.2: Architekturentscheid für Persistenz

Kapitel 7

Implementierung

7.1 JHipster Grundlagen

Die Basis des Web API Paver bildet eine Applikation, welche mit JHipster [1] generiert wurde. Dieses Framework erstellt eine Webapplikation mit Spring und Angular. JHipster verfügt über eine eigene Domain Language (JDL), mit welcher das Domain-Model abgebildet werden kann. Beim Importieren einer JDL-Datei in JHipster werden Entities für die Domänen-Objekte automatisiert erstellt, in dem JHipster von der Datenbanktabelle bis zur Angular-Komponente die nötigen Einrichtungen vornimmt [2].

Die Abbildung 6.1.3 zeigt den Aufbau einer mit JHipster-Applikation. Das Backend einer JHipster-App basiert auf Spring. Spring Boot bietet die Möglichkeit, eine produktionstaugliche Spring-Applikation mit minimalem Einrichtungsaufwand aufzusetzen. Das Grundprinzip von Spring Boot ist es, dem Benutzer die Einarbeit in Spring zu erleichtern, indem er ohne grosse Vorkentnisse eine funktionierende Umgebung einrichten kann [3]. Für die Authentifizierung und Autorisierung wird Spring Security verwendet. Spring MVC bietet die nötige Unterstützung für die Business Logic sowie ein REST API, über welches die Entitäten dem Frontend zur Verfügung gestellt werden. Für diese Kommunikation nutzt Spring Jackson als JSON-Prozessor. Um Daten zu persistieren verwendet Spring das Spring Data JPA. Im Frontend setzt JHipster auf Angular. Für generierte Entitäten stehen Views und Services als Angular-Komponenten zur Vefügung.

Zur Veranschaulichung der JHipster-Komponenten hat das Projektteam die Abbildung 7.1.1 erstellt.

[1] http://www.jhipster.tech/
[2] Raible, 2016 [10]: Generating Entities
[3] Long et al. 2017[9]: Chapter 1.2 - What Is Spring Boot?

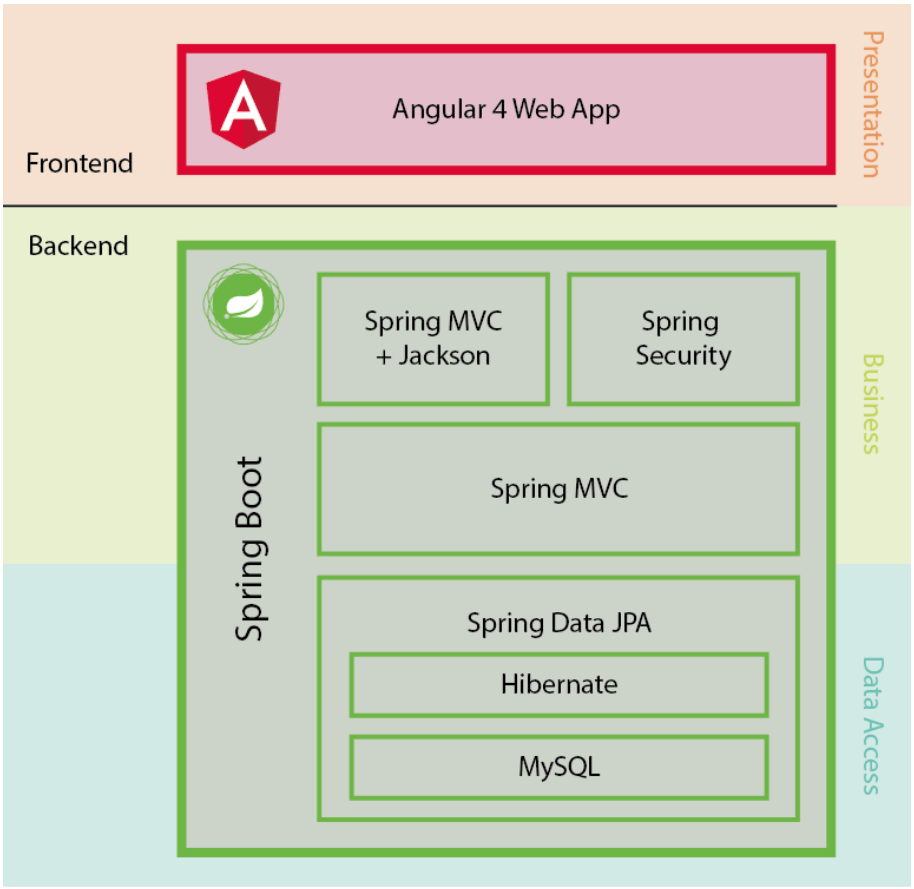


Abbildung 7.1.1: Technology Stack

7.2 Abhängigkeiten

Tabelle 7.2.1 listet Abhängigkeiten auf, welche explizit durch das Projektteam hinzugefügt wurden. Die Abhängigkeiten von JHipster können dem pom.xml und der package.json des Projektes entnommen werden.

Name	**Version**
JHipster (Framework)	4.9.0
jsoup	1.11.2
Swagger-Parser	1.0.31

Tabelle 7.2.1: Technische Abhängigkeiten

Die verwendete Version von JHipster entspricht dem Release zu Beginn der Bache-

lorarbeit (Septemnber 2017). Im Vergleich zum aktuellen Release 4.13 (Stand Dezember 2017) ist der verwendete Release veraltet, was zu Warnungen beim Build der Applikation führt.

7.3 Programmierschnittstellen

Die Kommunikation zwischen Frontend und Backend ist über eine Programmierschnittstelle realisiert. Über vordefinierte Methoden kann das Frontend Daten vom Backend abfragen oder dem Backend Daten übertragen. Die Schnittstelle zeigt auf, welche Daten an der Systemgrenze ausgetauscht werden können. Dieser Absatz basiert auf dem vom Projektteam entwickelten Web API. Die Beschreibung des API wurde mit Swagger[4] umgesetzt.

Dieses Swagger-Dokument enthält eine Übersicht der Ressourcen, die in der Web-API verfügbar sind. Die Operationen sind ein weiterer Bestandteil und zeigen auf, auf welche Ressourcen diese angewendet werden können. Zusätzlich spezifiziert sind die Parameter pro Operation, deren Namen und Typ.

Auf den nachfolgenden Seiten werden die spezifizierten Ressourcen vorgestellt und für einzelne Ressourcen die ausgetauschten Nutzdaten näher beschrieben.

7.3.1 API

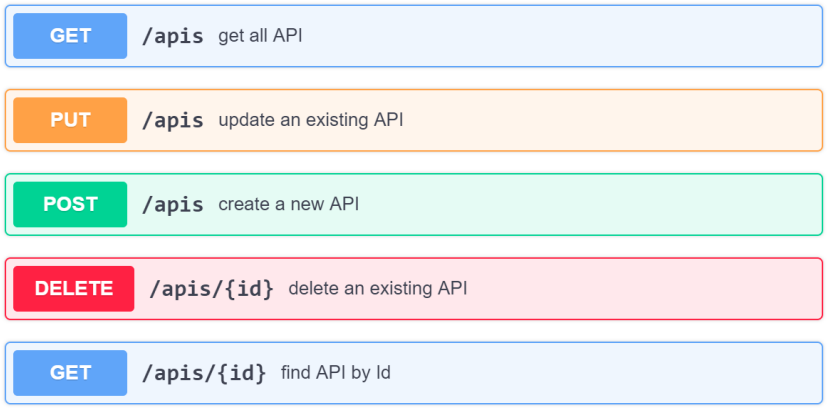


Abbildung 7.3.1: API Schnittstellen-Übersicht

In der Web-Applikation hat der Anwender die Möglichkeit ein API einzulesen oder selbst zu erstellen, dieses zu editieren und auch zu löschen. Mit `GET /apis` erhält

[4]https://editor.swagger.io

das Frontend vom Backend ein Array mit allen dem eingeloggten User zugeordneten APIs. Wird ein neues API erstellt, wir der eingeloggte Anwender automatisch als User abgespeichert. Die Nutzdaten sehen in der Beispielantwort von `GET /apis` wie in Abbildung 7.3.2 aufgezeigt, aus (die User Nutzdaten werden aus Platzgründen nicht angezeigt).

```
[
  {
   "id": 1,
   "description": "This is a sample Example Store",
   "title": "Example Store",
   "version": "1.0.0",
   "apiReferenceLinks": null,
   "user": {
     "id": 3,
       [User Fields]
   }
  }
 ]
```

Abbildung 7.3.2: API-JSON

Beim Erstellen eines neuen API können im dem Feld `apiReferenceLinks` URLs angegeben werden, welche als Basis für das Pattern Matching dienen.

7.3.2 Endpoint

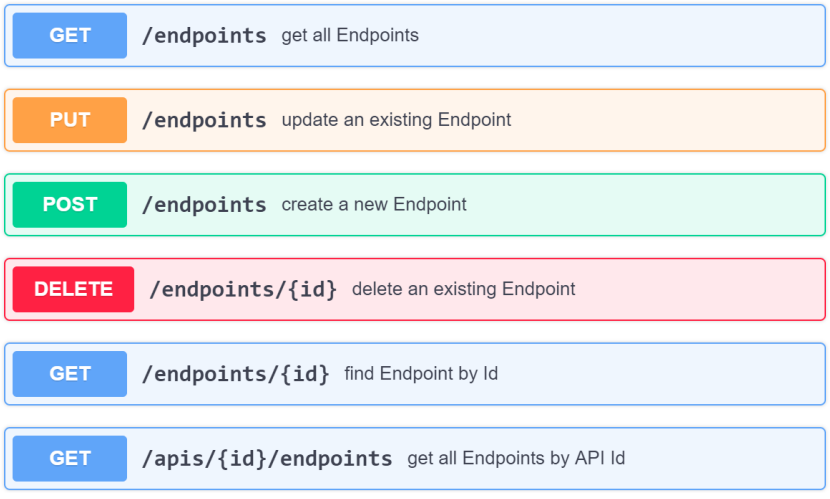


Abbildung 7.3.3: Endpoint Schnittstellen-Übersicht

Der Anwender hat die Möglichkeit einem eingelesenen oder selbst erstellten API ein oder mehrere Endpoints hinzu zu fügen. Ein anderer gängiger Begriff für Endpoint ist Ressource. Eine Ressource kann weiter editiert und gelöscht werden. Im Frontend wird `GET /endpoints` für die Anzeige aller Endpoints verwendet (relevant für Admin).

Über `GET /apis/{id}/endpoints` können alle Endpoints eines bestimmten API angefragt werden. Die Endpoint Nutzdaten, welche zwischen dem Backend und Frontend ausgetauscht werden, beinhalten die in Abbildung 7.3.4 aufgeführten Felder (mit Beispieldaten).

```
[
 {
   "id": 1,
   "name": "/estimates/price",
   "description": "Price estimation",
   "apiReferenceLinks": null,
    "api": {
     "id": 1,
     "description": "This is a sample Example Store",
     "title": "Example Store",
     "version": "1.0.0",
     "apiReferenceLinks": null,
     "user": {
        "id": 3,
         [User Fields]
      }
   }
]
```

Abbildung 7.3.4: Endpoint JSON

Aus den Nutzdaten kann entnommen werden, welchem API die Ressource zugeordnet ist.

7.3.3 Call (Request)

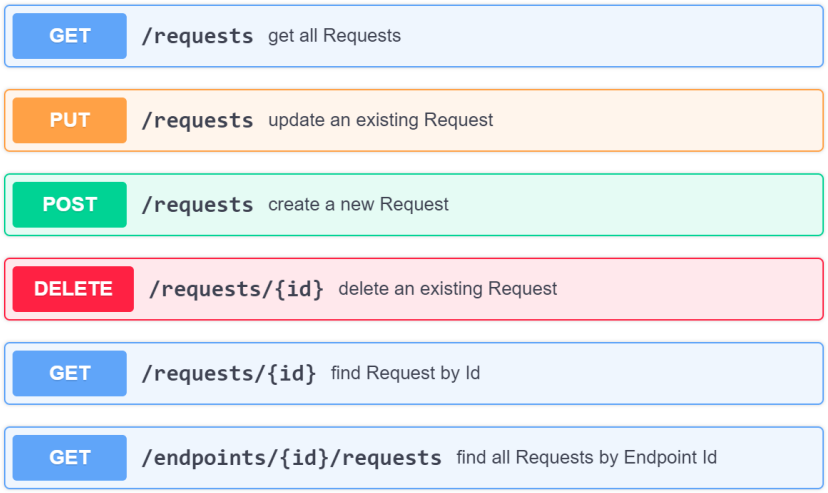


Abbildung 7.3.5: Request Schnittstellen-Übersicht

Einem Endpoint können ein oder mehrere Calls hinzufügt werden. Es ist hier anzumerken, dass im User Interface diese Ebene mit Call bezeichnet, im Backend hingegen mit Request. Ein Call kann im Frontend über `GET /request/{id}` abgerufen werden. Sind alle Calls eines bestimmten Endpoints nötig, können diese über die Schnittstelle `GET /endpoints/{id}/requests` bezogen werden. Der Anwender hat ebenfalls die Option ein Call zu löschen oder zu editieren. Die Nutzdaten beinhalten die in Abbildung 7.3.6 aufgeführten Felder (Beispiel eines JSON). Diese Felder werden für die Anzeige des Calls im User Interface benötigt.

```
[
 {
  "id": 1,
  "name": "GET",
  "description": "The Price Estimates endpoint returns an estimated price",
  "path": "/estimates/price",
  "apiReferenceLinks": null,
  "endpoint": {
      [Endpoint Fields]
  }
 }
]
```

Abbildung 7.3.6: Call JSON

Das Feld `endpoint` referenziert den Endpoint, zu welchem der Call gehört.

7.3.4 Reply

Abbildung 7.3.7: Reply Schnittstellen-Übersicht

Ein Reply kann ohne einen Call (Request) nicht existieren. Erstellt der Benutzer im Dokumentationsassistenten einen neuen Call, wird automatisch ein Reply miterzeugt. Im Frontend wird der Reply für die Zuordnung zum jeweiligen Call benötigt. Für die Kommunikation zwischen Backend und Frontend werden keine weiteren Felder benötigt.

Abbildung 7.3.8 zeigt die JSON-Struktur auf.

```
[
 {
  "id": 1,
  "hasRateLimit": "false",
  "request": {
    [Request Fields]
  }
 }
]
```

Abbildung 7.3.8: Reply JSON

Der Reply kann wiederum bearbeitet und vom Anwender gelöscht werden.

7.3.5 Parameter

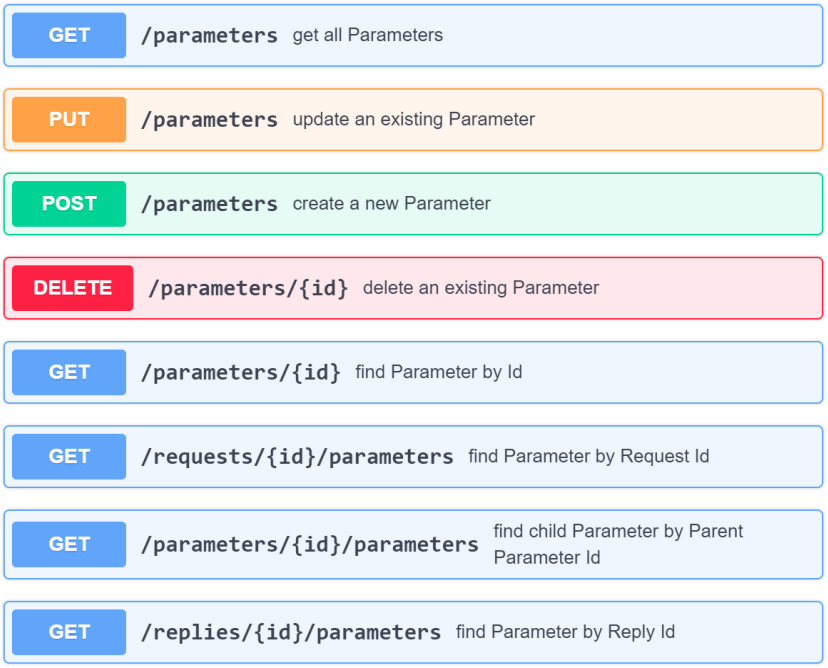


Abbildung 7.3.9: Parameter Schnittstellen-Übersicht

Ein Parameter wird entweder einem Reply oder einem Call zugeordnet. Das Frontend hat verschiedene Möglichkeiten, vom Backend die Parameter zu erhalten. Einerseits können über die Schnittstelle `GET /parameters` alle Parameter vom eingeloggten User abgefragt werden, andererseits nur die Parameter von einem Call (`GET /requests/{id}/parameters`) oder Reply (`GET /replies/{id}/parameters`).

Ein Parameter kann einen weiteren Parameter enthalten, so hat der Letztere in den Nutzdaten eine Referenz auf den Parent-Parameter. Im User Interface ist es für den Anwender möglich, ein Parameter neu zu erstellen, zu editieren oder zu löschen.

Abbildung 7.3.10 zeigt die JSON-Struktur mit Beispieldaten des Parameters auf, wie sie zwischen Backend und Frontend ausgetauscht werden.

```
[
 {
  "id": 1,
  "name": "id",
  "description": "",
  "valueType": "integer",
  "isPrimitive": "true",
  "parent": null,
  "reply": {
     [Reply Fields]
   },
  "request": null,
  "parent": {
     [Parent Parameter Fields]
  }
 }
 ]
```

Abbildung 7.3.10: Parameter JSON

7.3.6 Pattern Appearances

Einem API, Endpoint, Call, Reply und Parameter können die im Kapitel 3 aufgelisteten Patterns zugeordnet werden. Für jede Ebene besteht eine Tabelle mit den entsprechenden Patternzuordnungen. Will ein Benutzer des Dokumentationsassistentin im User Interface ein API bearbeiten und Patterns mittels Drag&Drop auf die API-Ebene ziehen, erfolgt ein entsprechender Eintrag in die Datenbank, mit ID des Patterns sowie ID des APIs. Das JSON zu den API Pattern Appearances sieht wie in Abbildung 7.3.11 aus.

```
[
 {
  "id": 1,
  "api": {
   [API Fields]
  }
  "pattern":  {
   [Pattern Fields]
  }
 }
]
```

Abbildung 7.3.11: API Pattern Appearance JSON

Die Schnittstellenbeschreibungen zu den Pattern Appearances können dem Schnitt-

stellenbeschreibung Pattern Appearances im Kapitel A, Seite 96, entnommen werden.

7.3.7 Pattern

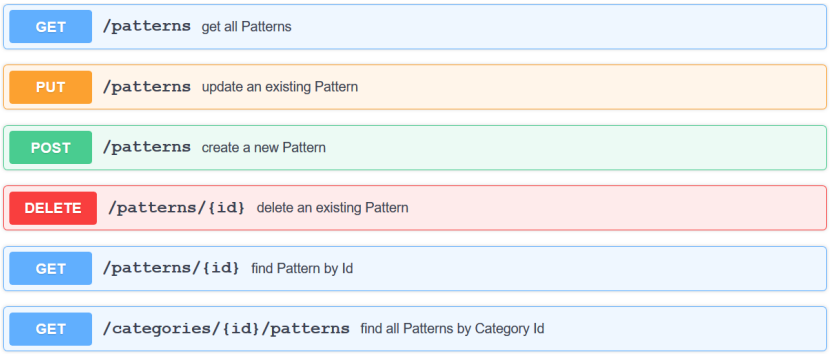


Abbildung 7.3.12: Pattern Schnittstellen-Übersicht

Für die Anzeige der Patterns-Icons im User Interface wird vom Frontend die Schnittstelle `GET /patterns` verwendet.

Die JSON Struktur kann aus der Abbildung 7.3.11 (mit Beispieldaten) entnommen werden.

```
[
  {
    "id": 1,
    "name": "Atomic Parameter",
    "description": "Atomic Parameter Description",
    "patternLink": "",
    "searchTerms": "single scalar representation, dot",
    "patternIcon": "PHN2ZyB4bWxuczOiaHR0c....",
    "patternIconContentType": "image/svg+xml",
    "constraint": "Parameter,NestedParameter,Request,Reply",
    "category": {
      [Category Fields]
    }
  }
]
```

Abbildung 7.3.13: Pattern JSON

Ein Pattern ist jeweils einer Pattern-Kategorie zugeordnet.

7.3.8 Category

Abbildung 7.3.14: Pattern Schnittstellen-Übersicht

Die Kategorien werden im User Interface angezeigt und somit besteht auch hier eine Schnittstelle zur Abfrage der Kategorien. Der Anwender kann die Kategorien in der Web-Applikation direkt anpassen und wenn nötig löschen.

7.4 Web-Applikation

Die nachfolgende Dokumentation beleuchtet wichtige Aspekte im Zusammenhang mit der Frontend-Entwicklung, indem sie die Überlegungen, welche zum aktuellen Erscheinungsbild und der Funktionalität des Web API Paver führten beschreibt.

7.4.1 Frontend Grundlagen

Die von JHipster generierte Angular App (Typescript) befindet sich im Ordner des „src/main/webapp“ und umfasst eine funktionsfähige Angular-Applikation, welche für den Prototyp erweitert wurde. Für ein besseres Verständnis des Aufbaus der Webapp werden in diesem Abschnitt die Grundlagen des Web API Pavers beschrieben.

Entities

Beim Generieren der Applikation erstellt JHipster für jede Entität ein Modul mit CRUD -Funktionalitäten, die als Services zur Verfügung stehen. Jede Entity verfügt über eine Service Klasse, die aus Angular-Funktionen für die eben erwähnten Funktionalitäten besteht. Diese Funktionen senden http Requests (GET, POST, PUT, DELETE) an das API des Backend und geben die JSON Antworten als konvertierte Objekte des entsprechenden Entity-Typs zurück.
Entitäten können über die JHipster Console hinzugefügt werden. Über den Navbar-Eintrag „Entities“ können die Entitäten in Tabellenform (sogenannte JHipster Cards) abgerufen und bearbeitet werden. Für Verwendung der App werden diese Darstellungen nicht benötigt, da auf dem Homescreen eine Single-Page App gestartet wird. Da diese Views für Verständnis- und Testzwecke jedoch durchaus nützlich sein können, bleiben sie auf der Benutzeroberfläche erhalten.

Event Manager

JHipster stellt einen eigenen Event Manager zur Verfügung. Die generierten Entities nutzen diesen um Änderungen mitzuteilen. Der `JhiEventManager` implementiert das Observer Pattern. Jede Komponente kann sich beim Event Manager registrieren, indem sie dem Manager eine Subscription mit Eventname und Callback angibt. Gleichzeitig können die Komponenten mit einem Event über den Broadcast des Managers registrierte Komponenten benachrichtigen. Die Daten, die übertragen werden sind im Event gespeichert. Jeder Observer der unter dem entsprechenden Eventnamen registriert ist, erhält das Event und somit die Daten. Dieser Broadcast ermöglicht eine einfache Kommunikation zwischen den Komponenten mit niedriger Kopplung.

Drag and Drop

In Hinsicht auf die Implementation der Drag and Drop Funktionalität ist es wichtig, die Unterschiede zwischen Browser (DOM) Events und Angular Events zu verstehen. Die zwei wichtigsten im Zusammenhang mit dieser Applikation sind folgende:

- Angular Events „bubbeln“ nicht wie DOM-Events. Während im DOM das Event jeweils bis zum Rootelement weitergeleitet wird (wenn es zuvor nicht explizit abgefangen wird), wird bei Angular der Event lediglich bis zum Parent weitergeleitet.
- Bei einem Drag Event des DOMs können während dem Drag-Vorgang keine Daten aus dem Event gelesen werden. Dies ist nur bei einem Drop Event möglich.

Um das Verhalten beim Drag und Drop so zu implementieren, dass einerseits die Daten des Events (auch während dem Draggen) korrekt gelesen und andererseits Events auf der richtigen Ebene abgefangen werden, verwendet die Applikation den Dragservice und die beiden Direktiven `Droptarget` und `Draggable`. Zusammen bilden diese Elemente eine Zwischenschicht, in der DOM-Events abgefangen werden und als Angular Event weitergegeben werden. Die Daten werden über die Dragservice-Komponente ausgetauscht.

UI Elemente, die draggable sein sollen, werden mit der Draggable-Direktive versehen. Mit dieser Direktive werden Zonen angegeben, auf welche das Element gezogen werden darf. Bei einem Drag-Vorgang werden die Dragevents über diese Direktive verarbeitet, die Daten des UI Elementes an den Dragservice weitergegeben und die entsprechenden Angular Events gefeuert.
Das Gegenstück bildet die Droptarget-Direktive. Sie wird auf UI Elemente angewendet, welche auf ein Drop Event reagieren sollen (wenn ein Draggable über das Element gezogen wird). Dem Target wird eine Zone zugewiesen, so dass es nur dann reagiert, wenn sich ein passendes Draggable über dem Target befindet.

7.4.2 Home-Komponente

Der Web API Paver verwendet die Home-Komponente von JHipster, die standartmässig aus einer simplen Seite besteht, als Einstiegspunkt der Single Page Applikation. Die JHipster Komponenten sind nach wie vor über die Navigationsleiste erreichbar, ihre Verwendung ist jedoch keine Voraussetzung.

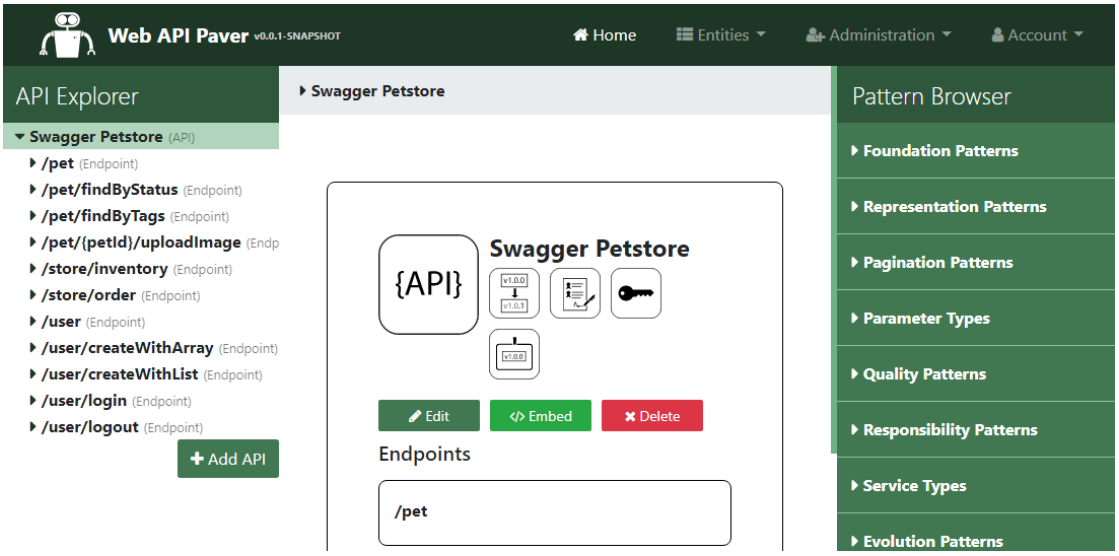


Abbildung 7.4.1: Home Komponente

Abbildung 7.4.1 zeigt die Benutzeroberfläche der Home-Seite. Sie ist unterteilt in drei Subkomponenten (von links nach rechts):

- API Explorer
- Visualisierung
- Pattern Browser

Die Darstellung mit der erwähnten Unterteilung folgt einem klaren Grundgedanken aus der Domainanalyse. Der API Explorer auf der linken Seite legt den Fokus auf das API, die Struktur wird übersichtlich dargestellt. Die rechte Seite zeigt die Patterns mit den zugehörigen Informationen, ohne Bezug auf ein spezifisches API. Diese beiden Bereiche zu verbinden ist die Kernaufgabe des Web API Pavers, daher befindet sich in der Mitte die Visualisierung, die genau das macht. Sie visualisiert das API auf eine interaktive Weise und ergänzt es durch Patterns die gefunden bzw. hinzugefügt wurden.

Die drei Bereiche sind logisch voneinander getrennt. Für jeden Bereich wurde eine eigene Angular Komponente erstellt. Diese drei Komponenten kommunizieren indirekt über den JHipster Event-Manager miteinander. Die einzelnen UI Elemente registrieren sich sobald sie erstellt wurden bei einer Entity- oder der einer andern UI-Komponente, um bei einer Änderung benachrichtig zu werden. Eine spezifischere Schnittstellenbeschreibung ist im nächsten Abschnitt bei der entsprechenden Komponente zu finden.

API Explorer

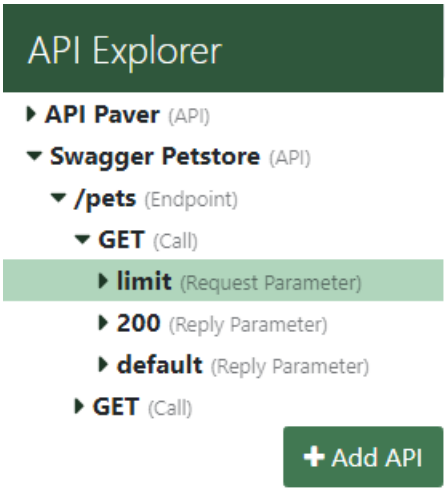


Abbildung 7.4.2: API Explorer

Der API Explorer erinnert stark an einen Explorer, wie man ihn von Betriebssystemen, IDEs oder anderen Applikationen kennt. Die Idee dabei ist, eine vertraute Navigationsmöglichkeit zu schaffen, die ohne weitere Erklärung klarkommt. Während die Hierarchie eines API nach dem einlesen sofort ersichtlich ist, war die Terminologie im Bezug auf die einzelnen Hierarchieebenen zu Beginn noch nicht geklärt. Da die Applikation mit den Begriffen API, Endpoint, Call, Parameter arbeitet, fiel daher die Entscheidung, auf jeder Ebene durch eine Beschriftung klar zu machen, was sich auf der entsprechenden Schicht befindet.

Aufbau

Die API-Explorer Komponente teilt sich in weitere Subkomponenten auf. Für jede Hierarchiestufe in der API Struktur gibt es eine Komponente, die an die entsprechende Entität geknüpft wird. Beim laden des API Explorers werden zuerst sämtliche APIs, die vom eingeloggten Benutzer erstellt wurden abgefragt. Für jedes API wird eine Komponente erstellt, in welche die jeweiligen Informationen gespeichert sind. Für jede Referenz zu einem Endpoint wird wiederum eine Subkomponente geladen, die eine Anfrage an den Server für die entsprechende Endpoint-Entity sendet. Dasselbe geschieht auf Call- wie auch auf Parameter-Ebene. Jeder Knoten bildet wiederum den Einstiegspunkt bzw. den Rootnode für die untergeordneten Komponenten.
Da Parameter verschachtelt werden können, wird ab dieser Ebene die weitere Struktur rekursiv aufgebaut, indem die Childnodes eines Parameters wiederum eine eigene Parameter-Tree-Komponente laden, in der sie als Root gesetzt sind.

Schnittstellen
Der API Explorer verfügt über Schnittstellen in zwei Bereichen. Einerseits kommuniziert er mit den Entities, andererseits soll mittels Benutzerinteraktion die Visualisierungskomponente benachrichtigt werden, damit die entsprechende Visualisierung dargestellt wird.

Der API Explorer kommuniziert mit folgenden Entities[5]:

API	Die API Service Komponente wird verwendet, um Informationen zu den APIs abzufragen und die Gliederungsebene im Explorer darzustellen. Des weitern wird über diesen Service ein eines API angelegt.
Endpoint	Die Endpoint Service Komponente wird verwendet, um Informationen zu den Endpoints abzufragen und die Gliederungsebene im Explorer darzustellen (Read Only).
Call	Die Call Service Komponente wird verwendet, um Informationen zu den Calls abzufragen und die Gliederungsebene im Explorer darzustellen (Read Only).
Parameter	Die Parameter Service Komponente wird verwendet, um Informationen zu den Parametern abzufragen und die Gliederungsebene im Explorer darzustellen (Read Only).

Tabelle 7.4.1: Übersicht der Schnittstellen im API Explorer

Die Entities aus der Tabelle 7.4.1 werden alles gleichbehandelt. Über die Service Klasse werden Calls an das Backend gesendet, um die Entity in JSON-Form zu erhalten. Die Service Klasse konvertiert das JSON in den entsprechenden Typescript-Klassentyp, um Typsicherheit zu garantieren. Das Objekt wird dem Template der jeweiligen Subkomponente des API-Explorers als Input mitgegeben.

Die Subkomponente registriert sich nach dem Instanziieren beim JHipster Event Manager für Events, die die entsprechende Entity betreffen. Bei CRUD-Operationen feuert die Entity-Komponente über den Event Manager Broadcast ein Event, das als Auslöser für die Funktion in der API Explorer Subkomponente dient.

Die Visualisierungskomponente kommuniziert mit der API Komponente ausschliesslich über den JHipster Event Manager in beide Richtungen. Bei einem Klick auf eine Ebene der API Struktur wird die Visualisierungskomponente benachrichtigt, die Darstellung anzupassen. Wenn in der Visualisierungskomponente navigiert wird (beispielsweise über die Breadcrumb Navigation oder durch Doppelklick auf eine Sub-

[5] siehe Absatz 7.3

komponente der dargestellten Ebene), sendet die Visualisierung ein Event, auf welches der API Explorer reagiert und die die jeweilige Ebene aufklappt und als ausgewählt markiert.

Pattern Browser

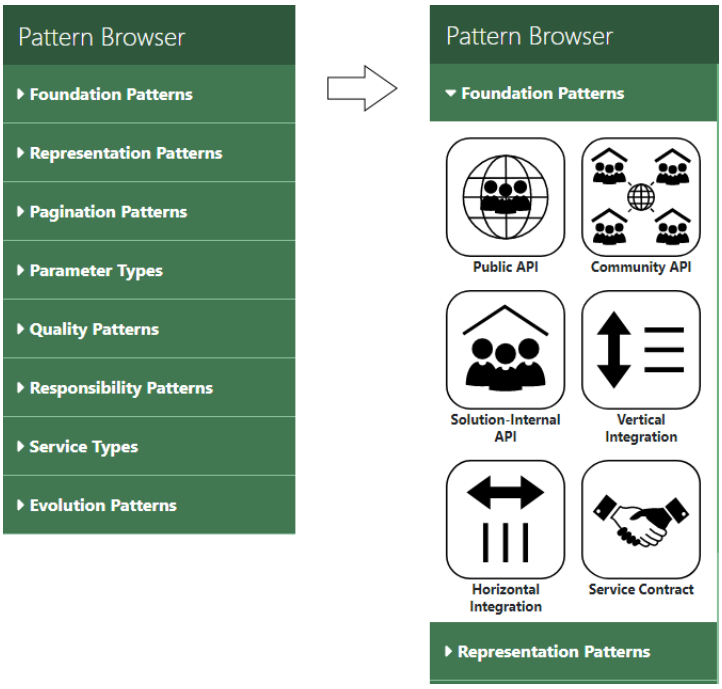


Abbildung 7.4.3: Pattern Browser

Aufgabe dieser Komponente ist es, die Patterns übersichtlich darzustellen um dem Benutzer die Auseinandersetzung mit der Patternsprache zu erleichtern. Die Sammlung der Patterns wird in Kategorien unterteilt dargestellt. Die Patternicon sind mit den zugehörigen Patternbeschreibung auf der Website[6] der Patternsprache verknüpft, wodurch der User durch einen Klick zu mehr Informationen kommt.

Aufbau

Der Pattern Browser stellt die Informationen der beiden Entity-Komponenten Pattern und Category dar. Die Komponente lädt bei der Initialisierung die Kategorien über die Service Klasse der Category Entity. Für jede Kategorie wir eine Subkomponente erstellt, welche die Patterns diese Kategorie laden und darstellen. Im HTML Template der Subkomponente werden die Patterns mit der Draggable-Direktive[7]

[6] Die Website ist noch nicht öffentlich zugänglich, daher wird eine Authentifizierung erfordert
[7] Siehe "Drag and Dropïm Abschnitt "Grundlagen Frontend"

versehen, um Drag and Drop zu ermöglichen. Als Dropzones ein Pattern werden die Constraints aus dem Datenfeld des Patternobjekts gesetzt. So wird über die Dragfunktionalität des Browsers verhindert, dass ein Pattern auf eine Ebene gezogen werden kann, für das es nicht zugelassen ist.

Schnittstellen
Die Schnittstellen des Pattern Browsers lassen sich ebenfalls in zwei Bereiche unterteilen. Einerseits wiederum die Schnittstelle zu den betroffenen Entities (Category und Pattern), andererseits die Kommunikation mit der Visualisierung. Letzteres verwendet Browser Events, wodurch wiederum eine lose Kopplung entsteht, diesmal aber nicht über den JHipster Event sondern über Drag and Drop Aktionen.

Der Pattern Browser kommuniziert mit folgenden Entities[8]:

Pattern	Über die Pattern Service Komponente werden Patterns über die ID der Kategorie abgerufen.
Category	Der Pattern Browser lädt über die Kategorie Service Komponente sämtliche Kategorien.

Tabelle 7.4.2: Übersicht der Schnittstellen im Pattern Browser

Für die Kommunikation zur Visualisierung werden Browserevents verwendet. Beim Start eines Dragevents (wenn der Anwender ein Pattern anklickt und wegzieht) wird das Pattern dem Event übergeben. Die Visualisierungskomponente reagiert auf Dragover und Drop Events. Informationen zum Pattern, welches gerade verschoben wird, können so über diese Events ausgelesen werden.[9]

[8] siehe Absatz 7.3

[9] Diese Beschreibung ist absichtlich abstrakt formuliert. Eine Zwischenschicht (der Dragservice und die zugehörigen Direktiven) ermöglichen es, Browserevents für Angular zu erweitern. Details dazu im Abschnitt "Drag and Drop" der Grundlagen.

Visualisierungskomponente

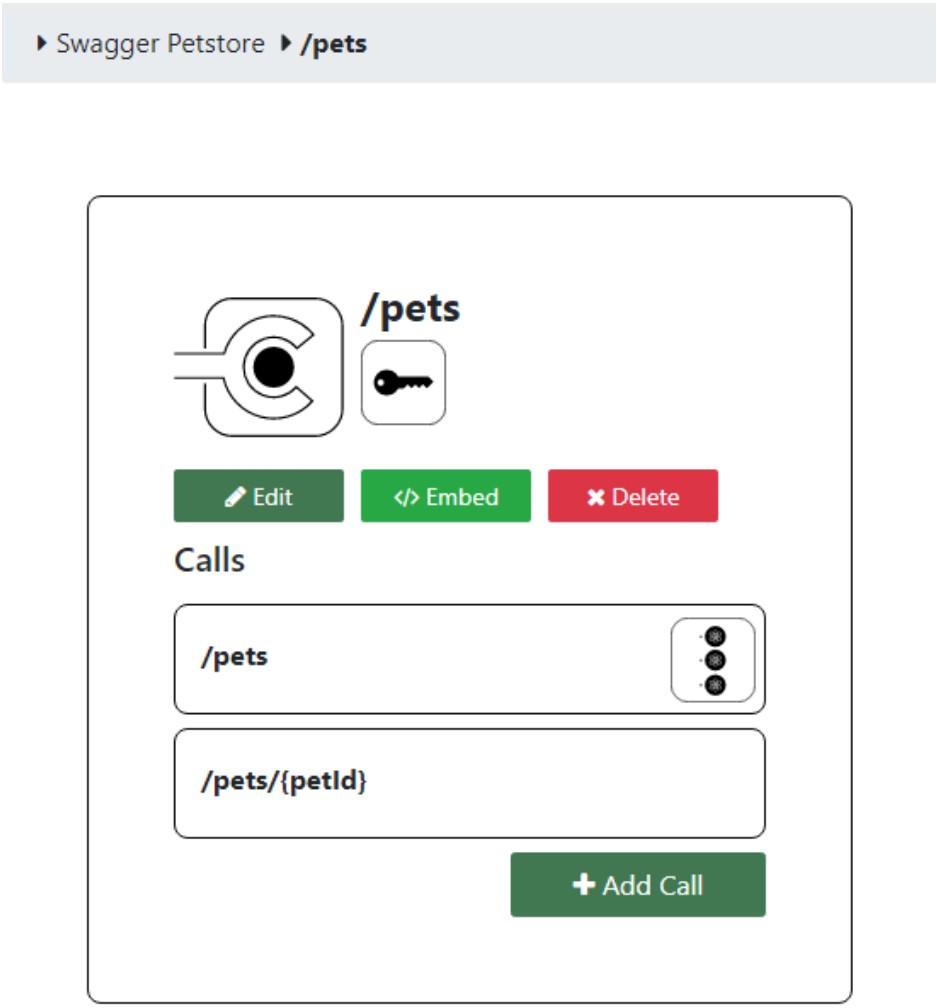


Abbildung 7.4.4: Visualisierungskomponente

Herzstück der Benutzeroberfläche ist die Visualisierungskomponente, die sich in der Mitte beider zuvor beschriebenen Hauptkomponenten befindet. Die Visualisierung stellt die ausgewählte Ebene des API Explorer dar und zeigt, welche Patterns zugeordnet wurden. Durch eine intuitive Benutzerinteraktion soll der Anwender durch sein API navigieren können und dabei, wenn nötig, Patterns und Ebenen ergänzen oder löschen können. Im Gegensatz zum API Explorer werden Inhalte detaillierter dargestellt, wobei sich der Benutzer auf einer bestimmten Ebene befindet. Die Beziehung zwischen API Explorer und der Visualisierungskomponente gleicht jener Beziehung der beiden Ansichten einer Master-Detail-View.

Aufbau
Die Visualisierungskomponente besteht aus zwei Teilen.

Breadcrumb Navigation:
Die Breadcrumb Navigation ist eine simple Aneinanderreihung von Links. Die Visualisierungskomponente speichert Name und Objekt der jeweiligen Ebene in einem Array. Bei einem Klick auf den Link wird dieselbe Funktion aufgerufen, welche für das „zeichnen“ der Visualisierung verwendet werden. Der Funktion wird als Parameter das entsprechende Objekt aus dem Array mitgegeben.

Visualisierung:
Auf jeder Ebene gibt es jeweils Angular Komponenten für die Visualisierung und für die Subvisualisierungen, wobei „Subvisualisierung“ für die Visualisierung der nachfolgenden Ebene steht. Da JHipster auf Code-Generierung setzt und somit für sämt/-liche Services eigene Klassen erstellt, ist es nicht für die Verwendung von Generics ausgelegt. Da die Umstellung auf Generics zur Folge hätte, das sämtliche autogenerierte Klassen überarbeitet werden müssen, hat sich das Projektteam gegen den Einsatz von Generics entschieden, auch wenn dies gerade in dieser Komponente von Vorteil gewesen wäre.

Entities werden nach dem selben Prinzip wie im API Explorer geladen. Für die Zuordnung zu den Patterns werden Pattern-Appearance-Entities verwendet. Zieht der Benutzer ein Pattern auf eine passende Ebene wird beim Drop-Event über die Service Klasse der Pattern-Appearance eine neue Zuordnung instanziert.

Schnittstellen
Die Visualisierungskomponente greift auf folgende Entities zu:

API	Die API Service Komponente wird verwendet um Informationen zu den APIs abzufragen und darzustellen. Des weiteren können über die Visualisierungskomponente APIs bearbeitet und gelöscht werden.
API Pattern Appearance	Über diese Komponente werden einem Element auf API Ebene Patterns zugeordnet.
Endpoint	Die Endpoint Service Komponente wird verwendet, um Informationen zu den Endpoints abzufragen und darzustellen. Endpoints können über die Visualisierungskomponente erstellt, bearbeitet und gelöscht werden.
Endpoint Pattern AppearanceKomponente	Über diese Komponente werden einem Element auf Endpoint Ebene Patterns zugeordnet.
Call	Die Call Service wird verwendet, um Informationen zu den Calls abzufragen und darzustellen. Calls können über die Visualisierungskomponente erstellt, bearbeitet und gelöscht werden.
Call Pattern Appearance	Über diese Komponente werden einem Element auf Call Ebene Patterns zugeordnet.
Parameter	Die Parameter Service Komponente wird verwendet, um Informationen zu Parametern abzufragen und darzustellen. Parameter können über die Visualisierungskomponente erstellt, bearbeitet und gelöscht werden.
Parameter Pattern Appearance	Über diese Komponente werden einem Element auf Parameter Ebene Patterns zugeordnet.

Tabelle 7.4.3: Übersicht der Schnittstellen der Visualisierungskomponente

7.4.3 Deployment

JHipster bietet einen Docker Support, indem beim Erstellen der Applikation ein Dockefile generiert wird, das in einem Container gestartet werden kann. Das Deployment in eine Cloud wird ebenfalls aktiv durch JHipster unterstützt. Da für dieses Projekt der Fokus auf der Entwicklung eines Prototyps lag, wurden diese Deployment-Möglichkeiten nicht ausreichend getestet und bleiben somit einer potentiellen Folgearbeit vorbehalten.

Kapitel 8

Pattern-Suche

8.1 Einlese- und Analysefunktion

Wird ein bestehendes Web API im OpenAPI-Specification Format[1] (nachfolgend „**OAS**“ genannt) über den Dokumentationsassistenten zur Analyse eingelesen, so wird das API auf Vorkommen von Patterns durchsucht und analysiert. Das Resultat ist eine visuelle Übersicht des APIs mit den zugeordneten Icons der gefundenen Patterns. Das Kapitel Patternsprache für Web API Design und Evolution auf Seite 5 zeigt die Pattern-Icons pro Kategorie auf.

In den nachfolgenden Absätzen wird auf die OAS und den Swagger-Parser eingegangen, sodann das Vorgehen beim Pattern Matching anhand von Beispieldaten näher erläutert.

8.1.1 OAS Version 2.0 und 3.0.0

In den Absätzen 2.1 und 2.2 im Kapitel Technische Grundlagen werden die Versionen 2.0 und 3.0.0 der OAS näher erläutert.

Der Dokumentationsassistent analysiert im derzeitigen Stadium API-Beschreibungen basierend auf OAS Version 2.0. Für die Analyse eines Web APIs im OAS Format wird der Swagger-Parser[2] eingesetzt. Für OAS Version 3.0.0 wurde der Support für die Swagger Core Java Library offiziell am 24. August 2017[5] bekanntgegeben. Das Announcement, mit dem dieser Support angekündigt wurde, sagt aus, dass es sich um einen Pre-Release handelt und sich an „Early Adopters“ richtet. Zum Start der Bachelorarbeit stand für OAS 3.0.0 der Swagger-Parser 2.0.0-rc0 (Pre-Release) zur Verfügung. Ende September 2017 wurde der zweite Pre-Release veröffentlicht

[1] https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md
[2] https://github.com/swagger-api/swagger-parser/releases

(v2.0.0-rc1), gefolgt vom dritten Pre-Release Mitte November 2017 (2.0.0-rc2)[6].

Aufgrund der Tatsache, dass die Unterstützung vom Swagger-Parser für OAS 3.0.0 anfangs der Bachelorarbeit erst in einer Pre-Release zur Verfügung stand, was sich bis zum heutige Zeitpunkt nicht geändert hat, wurde der Fokus auf OAS 2.0 gelegt. Für diese OAS Version steht ein stabiler Swagger-Parser zur Verfügung[6], was mit der Grund war, weshalb das Projektteam sich in erster Linie auf diese Version konzentriert hat.

8.1.2 Pattern Finding OAS 2.0

Ein Teil der Aufgabenstellung und Anforderung an die Web-Applikation ist es, dass der Benutzer ein Web API einlesen und analysieren lassen kann. Die Applikation analysiert das API auf vorkommende Patterns der Patternsprache. Für eine erfolgreiche Patternsuche und Implementierung der Analysefunktion ist es unumgänglich, die OpenAPI Specification[2] zu studieren und eine Zuordnung der Patterns gemäss Erläuterungen der Spezifikation vorzunehmen.

Wird ein Web API zur Analyse eingelesen, so werden Teile dieser API unterteilt in API, Endpoint, Call, Reply und Parameter. Die Patterns aus der Patternsprache werden jeweils einer oder mehreren dieser Ebenen zugeordnet. Ein Pattern kann je nach dessen Bedeutung und Funktionsweise nur einer oder mehreren dieser Ebenen zugewiesen werden; diese Einschränkungen sind pro Pattern in der Datenbank erfasst. So kann zum Beispiel das Public API Pattern aus der Kategorie der Foundation Patterns (siehe Tabelle 3.3.1) nur auf API-Level zugeordnet werden. Eine Übersicht dieser Einschränkungen kann dem Anhang B entnommen werden. Diese Einschränkungen können zu einem späteren Zeitpunkt in der Web-Applikation bei Bedarf angepasst werden.

Um dem Leser eine Idee dieser Analysetätigkeit zu geben, werden nachfolgend ein paar Ausschnitte aus diversen API-Spezifikationen abgebildet und aufgezeigt, wie die Patterns der Patternsprache daraus abgelesen werden können. Kapitel 8 der Studienarbeit von S.Kaslack und N.Dipner[12] enthält eine weiterführende Übersicht der Patterns und deren Vorkommen in öffentlich zugänglichen Web APIs.

Für Dokumentationszwecke werden die OAS Samples[3] oder das Petstore Sample[4] benutzt.

[3] https://github.com/OAI/OpenAPI-Specification/tree/master/examples/v2.0
[4] http://petstore.swagger.io/v2/swagger.json

Pattern-Suche auf API Ebene

Liest man ein Web API im OAS-Format über die Web-Applikation ein, wird durch den Swagger-Parser ein Swagger Objekt erstellt. Die Grafik in Abbildung 8.1.1 gibt Auskunft über die Root-Elemente beziehungsweise die Struktur eines Swagger Objektes. Die blau-markierten Felder sind für das Finden der Patternvorkommen auf API-Level von Bedeutung.

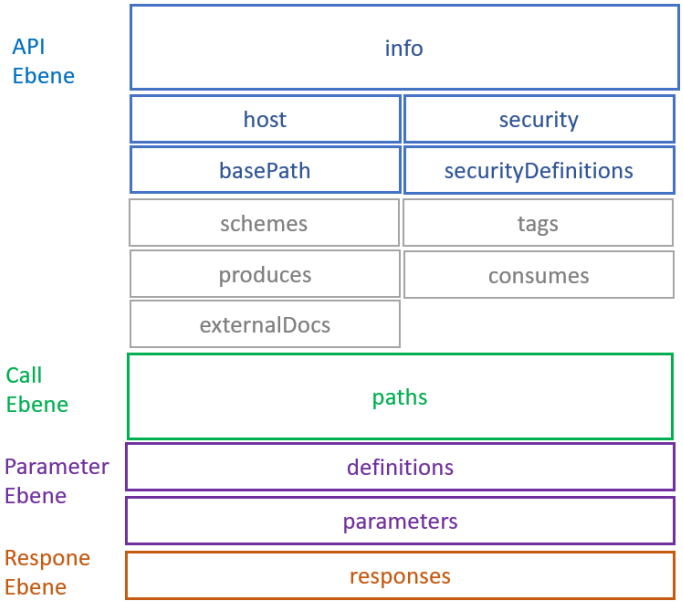


Abbildung 8.1.1: Struktur eines Swagger Objekts

Die oben erstellte Abbildung lehnt sich an die in der Spezifikation erwähnten `Root Document Objekte`[5].
Eine OAS Web API Beschreibung kann in YAML oder JSON geschrieben sein. Für die Analyse der Patternvorkommen auf der API Ebene werden nachfolgend die relevanten Felder im Swagger Objekt genauer betrachtet.

Abbildung 8.1.2 zeigt einen Auszug aus der Swagger Petstore OAS (in gekürzter Form).

[5]https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md#swagger-object

```
{
swagger: "2.0"
 info:
  description: "Sample description"
  version: "1.0.0"
  title: "Swagger Petstore"
  termsOfService: "http://swagger.io/terms/"
  contact:
   email:"apiteam@swagger.io"
  license:
   name: "Apache 2.0"
   url: "http://www.apache.org/licenses/LICENSE-2.0.html"
 host: "petstore.swagger.io"
 basePath: "/v2"
 securityDefinitions:
  api_key:
   type: "apiKey"
   name: "api_key"
   in: "header"
}
```

Abbildung 8.1.2: Auszug API Beschreibung API-Ebene

Aus der Abbildung 8.1.2 können bereits erste Patterns aus der Patternsprache dem API zugeordnet werden.

- `version:` Dies ist ein Pflichtfeld und gibt Auskunft über die Versionsnummer der entsprechenden Applikation. Während der Analyse wird untersucht, ob die Angabe der Versionsnummer der Beschreibung des **Semantic Versioning** Patterns entspricht (Versionierung mit mindestens zwei Stellen).

- `termsOfService:` Beinhaltet das Feld Angaben zu den Nutzungsbedingungen, findet das **Service Level Agreement** Pattern Anwendung.

- `securityDefinitions:` Im Beispiel ist als Authentisierungstyp apiKey angegeben, der global definiert ist. Dies entspricht dem Pattern **API Key** aus der Patternsprache.

- `basePath:` Dieses Feld entspricht dem Pfad, unter welchem das API zur Verfügung steht und ist relativ zum Host. Ist zum Beispiel `/v2` angegeben, so wird in der Analyse eine Zuordnung des **Version Identifier** Patterns vorgenommen.

Das obige Beispiel führt zu vier Patternzuordnungen auf API-Ebene:

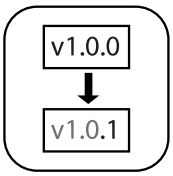
			
Semantic Versioning	Service Level Agreement	Version Identifier	API Key

Tabelle 8.1.1: Patternzuordnungen API-Ebene für Beispiel Web API

Pattern-Suche auf Endpoint Ebene

Auf dieser Ebene können auf Basis der OpenAPI Specifiation keine Patternzuweisungen vorgenommen werden. Der Strukturaufbau eines im OAS-Format erfassten APIs gibt keinen Aufschluss über vorkommende Patterns auf Endpoint Ebene.

Pattern Vorkommen auf Call Ebene

Wie im Absatz 7.3 Programmierschnittstellen erläutert, heisst im Backend diese Ebene Request. Ein Request gehört immer zu einem Endpoint und hat selbst immer ein Reply. Das grün-markierte Feld `paths` gemäss Abbildung 8.1.1 ist für das Finden der Patternvorkommen auf Call-Level von Bedeutung.

Abbildung 8.1.3 und Abbildung 8.1.4 zeigen einen zusammengesetzten Auszug aus dem Swagger Petstore OAS (in gekürzter Form), auf deren Basis die Pattenanalyse vorgenommen wird.

- `security:` Wird das Feld `security` für den Call explizit in der API-Beschreibung mit einem Wert angegeben, so werden die global gesetzten Security Definitionen überschrieben. Demzufolge kann das Pattern **API Key** diesem Call direkt zugeordnet werden.

- `/pet/findByTags:` Der Query Parameter ist ein Parameter-Typ, der oft in Web APIs anzutreffen ist. Diese erscheinen in der Anfrage-URL nach einem `(?)` und können mit einem `(&)` separiert werden. Der Aufruf für den in der Abbildung 8.1.3 definierten Query Parameter lautet gemäss Spezifikation `http://petstore.swagger.io/v2/pet/findByTags?tags=tag1&tags=tag2`. Die Analysefunktion überprüft, welchen Typ der Query Parameter hat. Ist dieser vom Typ `array` wird das Feld `items` überprüft. Ist im `items` Objekt kein

`enum` definiert, so wird gemäss der Patternbeschreibung in diesem Request das **Request Bundle** Pattern angewendet.

- `/pet/findByStatus:` Der Query Parameter `status` ist in diesem Beispiel vom Typ `array`. Gemäss der OAS 2.0 [2] muss das Feld `items` aufgrund dieses Typs (array) definiert werden. Ist der Typ von items ein `enum`, so sieht der Aufruf wie folgt aus:
`http://petstore.swagger.io/v2/pet/findByStatus?status=sold`. Der Query Parameter hat genau einen Wert. Gemäss der Patternbeschreibung findet in diesem Call das **Atomic Parameter List** Pattern Anwendung.

```
{
 swagger: "2.0"
  info:
   version: "1.0.0"
   title: "Swagger Petstore"
  host: "petstore.swagger.io"
  basePath: "/v2"
  schemes:
  - "http"
  paths:
   /pet/findByTags:
    get:
     summary: "Finds Pets by tags"
     description: "Finds Pets by tags"
     produces: [...]
     parameters:
     - name: "tags"
       in: "query"
       description: "Tags to filter by"
       required: true
       type: "array"
       items:
        type: "string"
       collectionFormat: "multi"
     responses:
      200:
       description: "successful operation"
       security:
        - petstore_auth:
        - "write:pets"
        - "read:pets"
}
```

Abbildung 8.1.3: Auszug API Beschreibung Call-Ebene /pet/findByTag

```
{
 ...
  paths:
   /pet/findByStatus:
    get:
     summary: "Finds Pets by status"
     produces: [...]
     parameters:
     - name: "status"
     in: "query"
     required: true
     type: "array"
     items:
      type: "string"
      enum:
      - "available"
      - "pending"
      - "sold"
    default: "available"
    collectionFormat: "multi"
...
}
```

Abbildung 8.1.4: Auszug API Beschreibung Call-Ebene /pet/findByStatus

Die Beispiel-Snippets aus der Abbildung 8.1.3 und Abbildung 8.1.4 haben drei Patternzuordnungen auf der Call-Ebene.

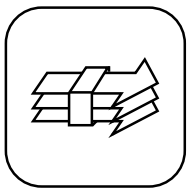
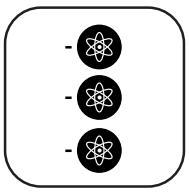
		
Request Bundle	Atomic Parameter List	API Key

Tabelle 8.1.2: Patternzuordnungen Call-Ebene für Beispiel Web API

Pattern-Suche auf Reply Ebene

Das `response` Objekt ist gemäss OAS ein Container für die erwarteten Antworten einer Anfrage. Es muss mindestens einen Response Code beinhalten. Empfohlen wird, den Response für eine erfolgreiche Operation anzugeben (200 HTTP Status Code). Das Feld `responses` (braun) gemäss Abbildung 8.1.1 ist für das Finden der Patternvorkommen auf Reply-Level von Bedeutung.

Die OAS gibt einem API Designer die Möglichkeit, via Header zusätzliche Informationen, die über das Resultat eines API Calls hinausgehen, zu erfassen. Ein API, das Rate Limit einsetzt, hat bei Verwendung der OAS die Möglichkeit, diese über die Response Headers zu erfassen.

Abbildung 8.1.5 zeigt eine Response mit einer „200“ HTTP Antwort. Das Beispiel stammt aus einem der Swagger OAS 2.0 Templates.

```
{
 responses:
  '200':
    description: An paged array of pets
    headers:
      X-Rate-Limit-Limit:
       description: The number of allowed requests in the current period
      type: integer
      X-Rate-Limit-Remaining:
       description: The number of remaining requests in the current period
       type: integer
      X-Rate-Limit-Reset:
       description: The number of seconds left in the current period
       type: integer
       schema:
        \$ref: '#/definitions/Pets'
      default:
       description: unexpected error
       schema:
        \$ref: '#/definitions/Error'
 ...
}
```

Abbildung 8.1.5: Auszug API Beschreibung Reply-Ebene

In OAS 2.0 ist das Setzen eines Rate Limits über das ganze API hinweg nicht möglich und muss je nach Call individuell definiert werden. Auf der Reply-Ebene können, basierend auf der OAS, nicht viele Patterns der Patternsprache gefunden werden. Dies liegt an dem Strukturaufbau von OAS und der Tatsache, dass nur ein Teil der Patterns der Patternsprache mit dieser Analyse gefunden werden können (siehe Absatz 8.1.3. für nähere Detailinformationen).

Pattern-Suche auf Parameter Ebene

Ein Parameter gehört zu einem Call oder Reply. Es ist möglich, dass ein Parameter verschachtelt ist und selbst weitere Parameter enthält. Für die Analyse der Parameter sind die in Abbildung 8.1.1 Felder `definitions` und `parameter` (violett) von Bedeutung. Die Patterns der Kategorie „Structural Representation Patterns“, aufgelistet in der Tabelle 3.3.2, finden hauptsächlich auf dieser Ebene Anwendung. Die Analysefunktion untersucht jeden Parameter und macht eine entsprechende Parameter-Pattern Zuordnung. Im Backend wird mittels der in Abbildung 8.1.6 aufgeführten Klassen überprüft, um welches Pattern es sich handelt.

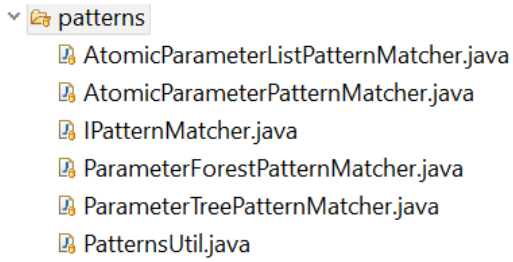


Abbildung 8.1.6: Struktur eines Swagger Objekts

Die nachfolgendenen Beispiele sollen dem Leser eine kurze Einleitung der „Structural Representation Patterns“ zum besseren Verständnis der nachfolgenden Analyseinformation geben.

Atomic Parameter Pattern

Es geht um den Austausch von simplen, unstrukturierten Daten. Wird zum Beispiel eine Anfrage auf eine bestimmte Ressource (`id`) gemacht, dann entspricht dieser `id` einem Atomic Parameter.

Atomic Parameter List Pattern

```
{
 id: 1,
 description: "This is a sample Store",
 title: "Example Store",
 version: "1.0.0",
 apiReferenceLinks: null,
}
```

Das JSON-Objekt enthält nur primitive Datentypen. Es geht um den Austausch von mehreren, simplen und unstrukturierten Daten. Die einzelnen Felder entsprechen dem Atomic Parameter Pattern.

Parameter Tree Pattern

```
[
 {
  id: 1,
  description: "This is a sample Store",
  title: "Example Store",
  version: "1.0.0",
  apiReferenceLinks: null,
  user: {
   id: 3,
   [User Fields]
  }
 }
]
```

Das JSON-Objekt hat nur einen Einstiegspunkt (Wurzel) und ein oder mehrere Datenstrukturen wie Tuples, Arrays, etc. In diesem Beispiel gibt es einen anonymen Einstiegspunkt (Objekt im Array).

Parameter Forest Pattern

```
{
 id: 1,
 description: "This is a sample Store",
 title: "Example Store",
 version: "1.0.0",
 apiReferenceLinks: null,
 user: {
   id: 3,
   [User Fields]
   }
}
```

Das JSON-Objekt hat keine Wurzel und beinhaltet nebst primitiven Datentypen Datenstrukturen wie Tuples, Arrays, etc.

OAS Parameter Spezifikation

Wird ein `POST` oder `PUT` Request an eine Web-Schnittstelle gesendet, so werden im Body der Nachricht oftmals die gleichen Felder gebraucht. Bei der API-Beschreibung möchte der API Designer diese gemeinsam benötigten Felder einmal setzen und nicht für jeden Call wiederholen. OAS bietet die Möglichkeit, hierfür `definitions` zu benutzen.

Nachfolgend werden Snippets aus dem Swagger Petstore[6] vorgestellt und gezeigt, welche Patternzuordnungen stattfinden.

[6] http://petstore.swagger.io/v2/swagger.json

```
{ ...
  get:
   description: ""
   parameters:
   - name: "username"
     in: "query"
     type: "string"
   - name: "password"
     in: "query"
     type: "string"
}
```

Abbildung 8.1.7: Patternzuordnung Parameter-Ebene/Atomic Parameter List

Die Query Parameter stellen einen **Atomic Parameter List** dar. Es sind zwei primitive Datentypen `string`, welche in der `URL` für diese `GET` Operation notwendig sind. Die einzelnen Felder wiederum sind **Atomic Parameters**.

```
{ ...
  post:
   parameters:
   - in: "body"
     name: "body"
     description: "Pet object to add"
     schema:
      \$ref: "#/definitions/Pet"
      ...
  definitions:
   Pet:
    type: "object"
    properties:
     id:
      type: "integer"
      format: "int64"
    category:
     \$ref: "#/definitions/Category"
    name:
     type: "string"
     example: "doggie"
}
```

Abbildung 8.1.8: Patternzuordnung Parameter-Ebene/Parameter Forest

Der hier definierte Parameter namens `body` ist vom Typ Body Parameter und somit ein Parameter, welcher dem Call zugeordnet ist. Wird eine `POST` Operation gemacht, um ein neues Element hinzuzufügen, sind die Felder gemäss der referenzierten Definition beim Erstellen des Elements mitanzugeben, sofern nötig. Die Struktur zeigt auf, dass die Felder in einem Objekt eingepackt sind und dass nicht nur primitive Datentypen in der `Pet Definition` vorkommen. Somit lässt sich aussagen, dass dies den **Parameter Forest** Pattern darstellt.

```
{
 /pet/findByTags:
  get:
   parameters:
   - name: "tags"
     in: "query"
     description: "Tags to filter by"
     type: "array"
     items:
     type: "string"
   collectionFormat: "multi"
   responses:
    200:
     description: "successful operation"
     schema:
      type: "array"
      items:
       \$ref: "#/definitions/Pet"
}
```

In der Antwort wird ein „200“ HTTP Response Code erwartet. Ist die Operation erfolgreich, so kommt die Antwort in einem `type array` mit Pet-Objekten. Gemäss obiger Definition zum Parameter Tree, handelt es sich hier um das `Parameter Tree` Pattern.

Abbildung 8.1.9: Patternzuordnung Parameter-Ebene/Parameter Tree

Die oben aufgeführten Beispiele von API-Beschreibungen sind nicht abschliessend und sollen einen Überblick über das Finden der „Structural Representation Patterns“ geben.

8.1.3 Übersicht der gefundenen Patterns über die Einlese-/Analyse-funktion

Im vorhergehenden Absatz wurde aufgezeigt, wie die Pattern-Suche in OAS-spezifischen APIs funktioniert. Ein Vergleich mit den im Kapitel 3 Patternsprache für Web API Design und Evolution aufgelisteten Patterns zeigt auf, dass in OAS-basierenden API-Beschreibungen nicht alle Patterns der Patternsprache „Web API Design and Evolution“ gefunden werden können.

Die sieben Patterns der Kategorie „Evolution“ beschäftigen sich mit dem Lifecycle Management von APIs und beinhalten Aspekte der Versionierung. Fragestellungen wie (i) sollen dem API-Consumer jeweils nur eine oder mehrere Versionen eines APIs zur Verfügung gestellt werden, (ii) soll für ein API eine limitierte oder lebenslange Garantie gegeben werden oder (iii) bietet man einem API-Consumer eine Beta-Version einer neuer Version an, werden in dieser Patterkategorie abgedeckt[13]. Die OpenAPI Specification gibt jedoch nur Auskunft über die Versionierung und op-

tional über den Version Identifier.

Die sechs Patterns der Kategorie „Foundation“ beschäftigen sich mit Design und Management von APIs. Die Patterns geben Auskunft über die Art, wie auf ein API zugegriffen wird, die Integration und den Service Contract[13]. Die OpenAPI Specification beinhaltet diese Aspekte nicht.

Die zehn Patterns der Kategorie „Quality“ geben Auskunft zu Qualitätsaspekten eines API. Mögliche Fragestellungen dieser Kategorie sind, (i) wie sollen APIs vor Missbrauch geschützt werden, (ii) wie ist es möglich, Latenzzeiten zu vermindern und (iii) soll ein Service Level Agreement Anwendung finden[13]. Die OpenAPI Specification bildet ein Teil dieser Fragestellungen in ihrem API Design Konzept ab.

Die Patterns der Kategorie „Responsibility“ beschäftigen sich mit semantischen Aspekten in einem API Design. Mögliche Fragestellungen dieser Kategorie sind, (i) um was für Daten handelt es sich - sind es Daten, welche sich selten oder häufig ändern, (ii) werden Daten als Referenz oder als ganzes Objekt zurückgegeben, (iii) welche Parametertypen werden unterschieden und (iv) wie erfolgt die Unterscheidung von verschiedenen Arten von Metadaten. Es ist schwierig, anhand der Informationen aus der OpenAPI Specification konkrete und richtige Patternzuordnungen zu machen. Die Parametertypen können gut unterschieden werden, Metadaten sind in der Spezifikation nicht weiter erläutert.

Die Tabelle 8.1.3 zeigt die Patterns auf, welche mit dem Dokumentationsassistenten über die Einlese-/Analysefunktion gefunden werden:

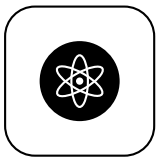
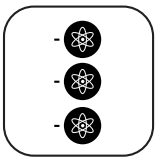
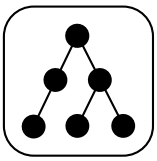
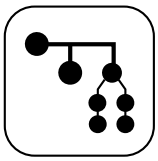
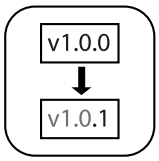
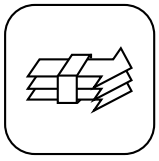
 Atomic Parameter	 Atomic Parameter List	 Parameter Tree	 Parameter Forest	 Semantic Versioning
 Version Identifier	 API Key	 Rate Limit	 Id Parameter	 Service Level Agreement
 Request Bundle				

Tabelle 8.1.3: Anwendbare Patterns auf Ebene der Einlese-/Analysefunktion

8.2 Freitextsuche

Der Dokumentationsassistent bietet einem Anwender die Möglichkeit, eine API-Beschreibung selber von Grund auf neu zu gestalten und in visueller Form Dritten zur Verfügung zu stellen. Steht dem API-Designer, der den Assistenten benutzt, seine API-Beschreibung im HTML-Format zur Verfügung, so hat er die Möglichkeit, bei der Erstellung eines neuen APIs eine oder mehrere URLs anzugeben. Die URLs werden im Backend durch eine Suchfunktion auf Vorkommen von Patterns der Patternsprache durchsucht. Es ist möglich, diese URLs zu einem späteren Zeitpunkt zu editieren und anzupassen. Dieselbe Funktionalität steht dem Anwender auf den Ebenen `Endpoint` und `Call` zur Verfügung.

Findet die Suchfunktion einen Match, wird die Visualisierung mit den entsprechenden Pattern-Icons ergänzt.

Damit die Pattern-Suche funktioniert, müssen vorab Suchwörter für die Patterns definiert und in der Datenbank gespeichert sein. Die Tabelle 8.2.1 zeigt ein Beispiel von Suchwörterzuordnungen für zwei ausgewählte Patterns:

Nr	Patternname	Suchwörter
1	Rate Limit	rate limit, speed limit, quota, usage limitation
2	API Key	key, OAuth, Authentication, limit usage, token, identification, Access Token, Shared Secret

Tabelle 8.2.1: Anzeige der Suchwörter für zwei ausgewählte Patterns

Die Suchwörter können über die Benutzeroberfläche des Dokumentationsassistents angepasst und ergänzt werden.

8.2.1 Hinweis zur Implementierung

Für die Textmenge, die es zu untersuchen gilt, genügt eine einfache Suche. Es wird über den Text iteriert und geprüft, ob ein Match mit dem ersten Buchstaben stattfindet. Wenn dies der Fall ist, wird der Rest des Wortes mit dem Text verglichen.

Beispiel 1: Im Dokumentationsassistent wird ein neues API erstellt mit den folgenden Daten:

- Title: GitHub API
- Description: Description GitHub API
- Version: 3.1.2
- Api Reference Links: https://developer.github.com/v3/, https://developer.github.com/v4/

Die Suchfunktion liefert jene Patterns zurück, welche für die untersuchten URLs gefunden wurden, unter Rücksichtnahme der im Pattern Constraints definierten Constraints.
Die Abbildung 8.2.1 zeigt die Pattern-Zuordnung auf API-Ebene.

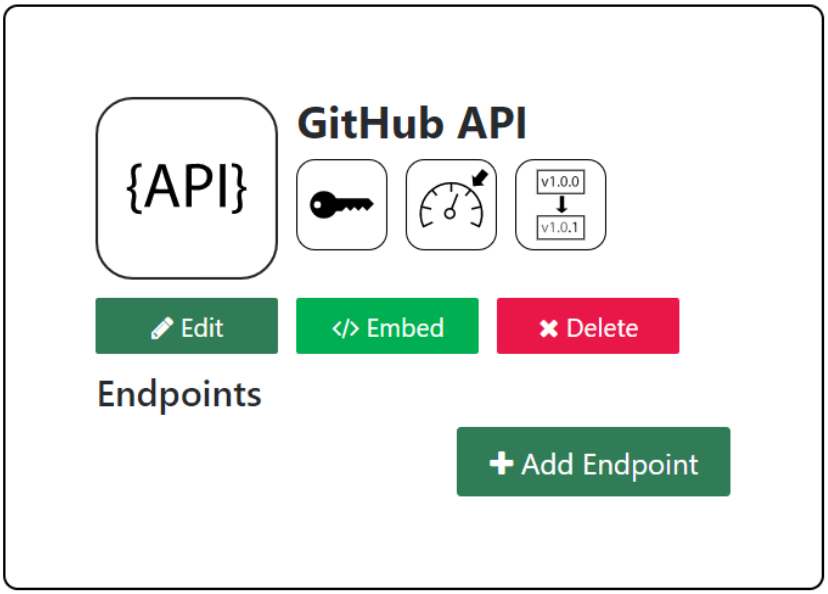


Abbildung 8.2.1: API-Patternzuordnung nach URL-Analyse

Analysiert man diese Suche mit der `Java.lang.System.nanoTime()` Methode, so wird für diese Analyse `0.31994` Sekunden gebraucht.

Beispiel 2: Annahme:„API hinzufügen“ wurde erfolgreich durchgeführt. Dem API soll jetzt ein Endpoint hinzugefügt werden. Beim Hinzufügen des Endpoints sollen externe Endpoint-Referenzen eines APIs als URLs angegeben werden. Der Endpoint wird wie folgt unter Berücksichtigung der in namerefsec:anhang2 definierten Constraints erstellt.

- Name: /payments/payment
- Description: Description GitHub API
- Api Reference Links: https://developer.paypal.com/docs/api/overview/ /

Die gefundenen Pattern sind in Abbildung 8.2.2 abgebildet. Die gefunden Patterns basieren auch auf den in Pattern Constraints definierten Constraints.

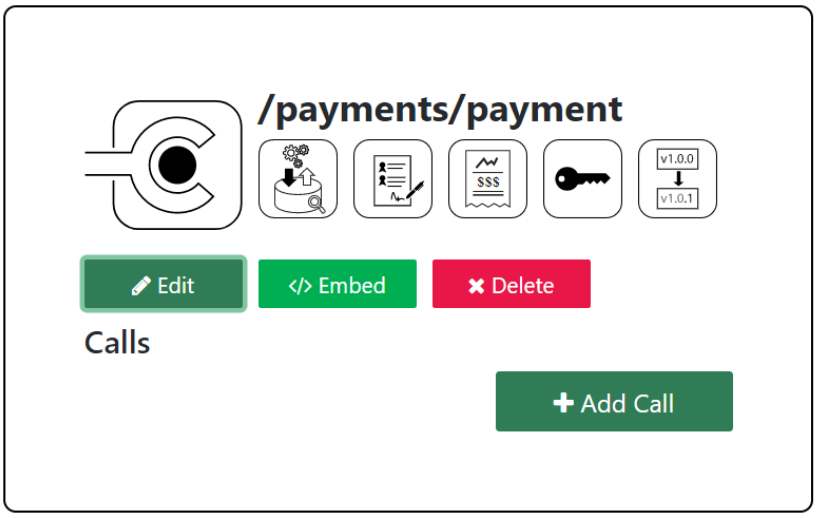


Abbildung 8.2.2: Endpoint-Patternzuordnung nach URL-Analyse

Das Erstellen des Endpoints und das Auffinden der Pattern dauert gemäss der Messung 1.6527 Sekunden.

8.3 Pattern-Anpassungen

Die in der Web-Applikation ersichtlichen Patterns bzw. Pattern-Icons können über den Dokumentationsassistenten angepasst werden. Folglich können Suchwörter, Einschränkungen (Constraints), Beschreibungen, URLs editiert oder das Icon ersetzt werden.

Das Pattern-Matching wird anhand der Patternnamen, wie sie in der Datenbank erfasst sind, vorgenommen. Aus diesem Grund ist es wichtig, folgende Regelung zu beachten:

Wichtiger Hinweis: Ändert sich der Patternname, so muss zusätzlich eine Anpassung des im Anpassungen von Pattern- und Kategorie-Bezeichnungen erwähnten Konfigurationsfiles vorgenommen werden, damit die Pattern-Suche für die Einlese- und Analysefunktion weiterhin erfolgreich funktioniert. Dies gilt ebenso für die Anpassung der Kategorie-Bezeichnung.

Kapitel 9

Tests

9.1 User Interface & Usability Testing

9.1.1 Phasen

Ein Projektablauf enthält typischerweise die folgenden fünf Phasen: Analyse, Konzeption, Design, Implementierung und Dokumentation. Die Konzeptionsphase kann in Grob- und Detailkonzeption aufgeteilt werden. Ein Usability Test kann in beiden Phasen durchgeführt werden, in der Grobkonzeption in Form von Scribbles oder einem interaktiven Prototyp und in der Detailkonzeption in Form des weiterentwickelten Prototyps oder eines programmierten Prototyps [7].

Nachfolgend werden die mit dem User Interface zusammenhängenden Testarten, welche während des Projektverlaufs vorgenommen wurden, näher erläutert.

9.1.2 Interaktive Wireframes

In der Elaborations-Phase hat das Projektteam Website-Gestaltungsentwürfe in Form von Wireframes ausgearbeitet und verschiedene Prototyping-Tools evaluiert. Für die Umsetzung wurde Wireframing-Tool „Adobe Experience Design CC“ gewählt (siehe Absatz 9.1.5 für den Evaluationsbericht).

Ziel war es, eine grobe Darstellung der Weblösung in der frühen Projektphase auszuarbeiten, die Anordnung der wichtigsten Inhaltselemente festzulegen sowie eine Idee zur Benutzerführung umzusetzen. Für das Design der Wireframes wurden die User Stories 4 als Grundlage herbeigezogen, um sicher zu stellen, dass die Gestaltungsentwürfe alle Funktionalitäten abbilden.

Die Wireframes wurden dem Auftraggeber und anschliessend einem weiteren Patternautoren präsentiert. Dabei ging es in dieser frühen Projektphase nicht um De-

tailfragen der Funktionalitäten, sondern um Inhaltselemente, den Strukturaufbau und die Rückmeldung zu den Designentwürfen. Die dabei entstandenen Diskussionspunkte und Anregungen zum Design sind in die nächsten Design- bzw. Prototyp-Entwürfe miteinbezogen worden.

In der Abbildung 9.1.1 und Abbildung 9.1.2 sind die ersten User Interface Entwürfe in Form von Wireframe-Skizzen abgebildet.

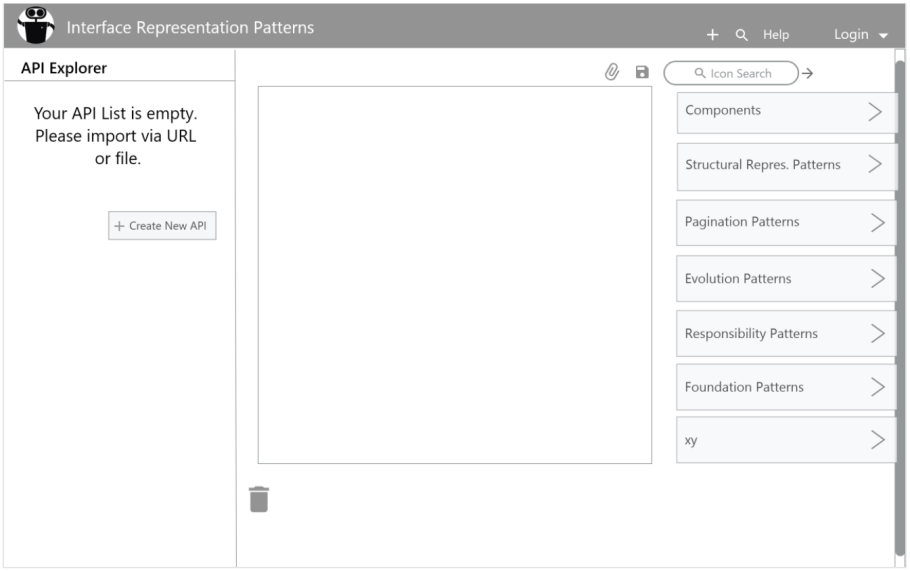


Abbildung 9.1.1: Wireframe Startseite

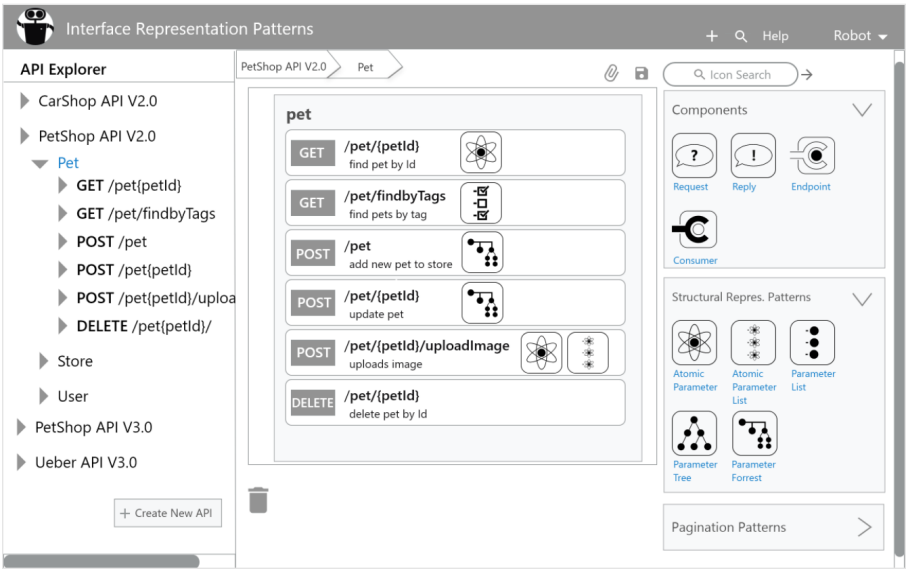


Abbildung 9.1.2: Wireframe Endpoint Ebene

Mit Hilfe dieser Wireframes konnten bereits erste Tests mit Bezug auf das User Interface und die Benutzerführung durchgeführt werden, was für das Projektteam die nachfolgende Arbeit zur Prototyp-Implementierung wesentlich vereinfacht hat.

9.1.3 UI-Walkthrough

Im letzten Drittel der Projektphase hat das User Interface mitsamt den implementierten Funktionalitäten mehrere Testphasen durchlaufen.

In erster Linie wurde der Prototyp mit dem Auftraggeber und einem Patternautor in unterschiedlichen Sitzungen auf die in der Applikation vorkommenden Terminologien überprüft, die Anordnung der Pattern-Icons kontrolliert und fehlende pattern-bezogene Informationen diskutiert. Das Projektteam hat diese Art von Test als „User Interface-Walkthrough“ (nachfolgend „UI-Walkthrough“) bezeichnet. Der Zweck bestand darin, das ganze User Interface und die verschiedenen Navigationsmöglichkeiten zu durchlaufen und sicherzustellen, dass alle wesentlichen Aspekte Bestandteil der Web-Applikation sind. Das Ergebnis der UI-Walkthroughs führte zu einer Liste von Umsetzungs- bzw. Anpassungswünschen, welche vom Projektteam für die weitere Implementierung miteinbezogen wurde. Die wesentlichen Punkte waren:

- Textuelle Anpassungen
- Pattern-Anpassungen (Beschreibung, Anordnung, Links, etc.)
- Anpassungen auf Visualisierungsebene (zusätzliche Buttons)

Ein wichtiger Aspekt des UI-Walkthroughs war der Austausch mit dem Auftraggeber sowie die Bestätigung, dass das vorgestellte Konzept bzw. UI-Design auch im Interesse des Auftraggebers ist. Ein anderer Vorteil, den man aus solchen Sitzungen hat, ist die kontinuierliche Verbesserung der Benutzerfreundlichkeit der Web-Applikation und Sicherstellung, dass zukünftige Benutzer mit der Anwendung keine negativen Überraschungen erleben.

Die Abbildung 9.1.3 zeigt das User Interface vor den UI-Walkthroughs auf. Die Abbildung 9.1.4 beinhaltet die vorgenommenen Anpassungen.

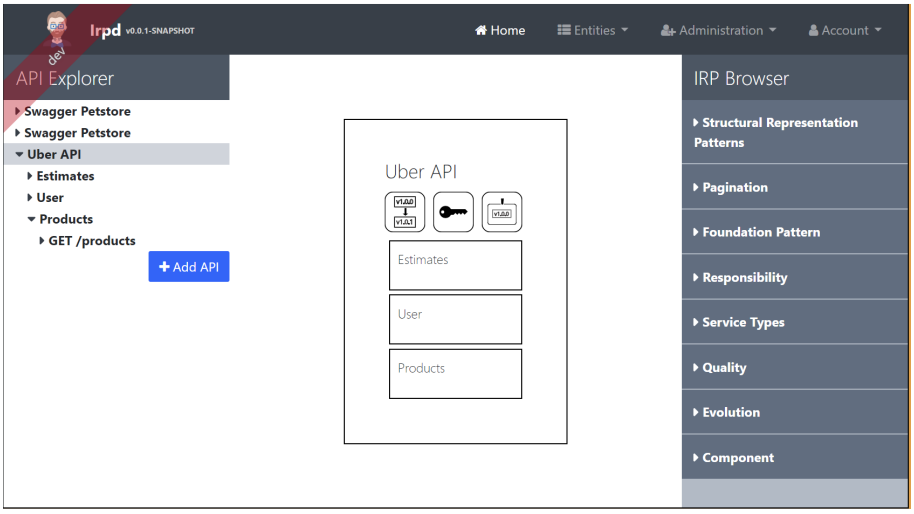


Abbildung 9.1.3: User Interface vor UI-Walkthrough

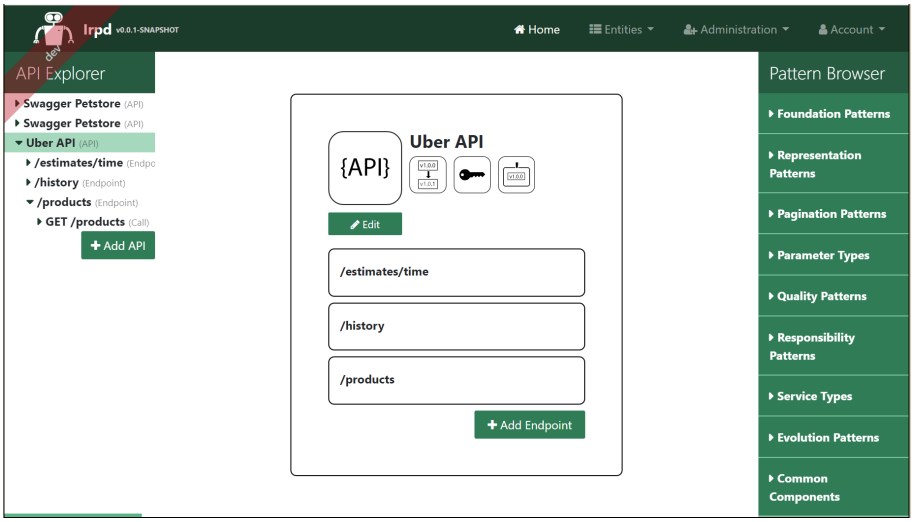


Abbildung 9.1.4: User Interface nach UI-Walkthrough

9.1.4 Usability Tests

Mit den Usability-Tests wurde die Web-Applikation auf Bedienbarkeit, Benutzerfreundlichkeit, Funktionsumfang, Struktur, Navigation und Inhalt getestet. Diese Art der qualitativen Befragungsmethode benötigt mehr Aufwand zur erfolgreichen Durchführung der Tests. Nebst dem Testkonzept müssen Entscheidungen getroffen werden, wie zum Beispiel (i) welche Personen für einen Usability-Test in Frage kommen, (ii) welche Vorbereitungsarbeiten notwendig sind, (iii) Durchführen der Tests

und (iv) Auswertung der Tests. Das Testkonzept diente in diesem Projekt zugleich als Befragungskatalog. Daraus sollten Fragen beantwortet werden, ob einzelne Teil gut auffindbar sind, Abläufe, Texte und Begrifflichkeiten verständlich sind und wie nützlich die Anwendung ist.

Das Projektteam hat insgesamt zwei Usability Tests in Form einer persönlicher Befragung durchgeführt. Die Testpersonen konnten die reale Applikation basierend auf den Fragekatalog testen bzw. bedienen und dem Projektteam zu jedem Schritt eine Rückmeldung geben. Die Auswertung der Usability Tests zeigte folgende Wünsche auf:

- Mehr Erklärungstexte im User Interface
- Pattern-Namen und Pattern-Beschreibungen hervorheben bzw. anzeigen
- Löschfunktionen deutlicher kennzeichnen
- Bessere Navigationsmöglichkeit für den Benutzer
- Anpassungen von ein paar wenigen Begrifflichkeiten

Das User Interface wurde aufgrund der Testauswertungen leicht angepasst. Aus der Abbildung 9.1.5 kann entnommen werden, dass wenn man mit der Maus über ein Pattern-Icon fährt, der Name des Patterns und die Kategoriezugehörigkeit angezeigt wird. Mit einem Doppelklick kommt man auf die entsprechenden API-Beschreibungsseiten. Die Löschfunktion ist für den Anwender neu sofort ersichtlich.

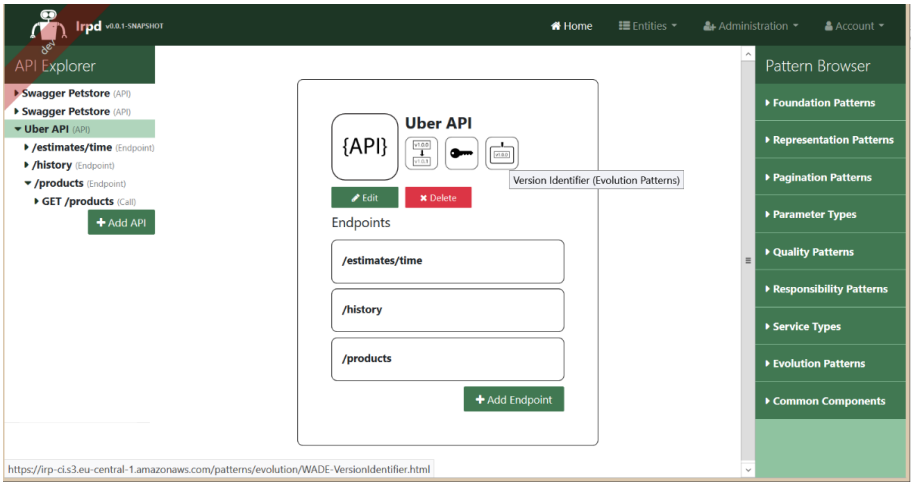


Abbildung 9.1.5: User Interface nach Usability-Tests

9.1.5 Auswahl Prototyping-Werkzeug

Für die Evaluation des Prototyping-Tools standen die nachfolgend aufgeführten Software-Prototyp-Lösungen zur Auswahl.

- Axure[1]
- Balsamiq[2]
- Adobe Experience Design CC[3],

Die Voraussetzungen für die in die engere Wahl miteinbezogenen Prototyping-Tools waren einerseits, dass es auf dem Windows10 Betriebssystem laufen muss, und dass das Projektteam mit dem Tool bereits zuvor gearbeitet hat (um die Einarbeitungszeit zu minimieren).

Axure

Axure ist ein Prototyping Werkzeug, welches viele mitinstallierten Widgets enthält, die einem Benutzer es ermöglichen, ein komplexes und interaktives Wireframe zu erstellen. Das Verstehen des ganzen Lieferumfangs und deren Funktionalitäten können für einen Benutzer anfänglich eine Hürde darstellen.

Vorteile
- Gratisversion für Studenten
- interaktive Wireframes
- gute Dokumentation und Tutorials
- viele Widgets mit integriert

Nachteile
- komplizierter Aufbau
- längere Einarbeitungszeit

Balsamiq

Balsamiq bietet als Prototyping-Tool einen schnellen und einfachen Einstieg in die Welt der Wireframes und ist gut geeignet für Low-Level Wireframes. Komplexe und interaktive Wireframe-Designs sind mit diesem Tool nicht umzusetzen.

Vorteile
- einfache Handhabung
- Gratisversion für HSR-Studenten
- gute Dokumentation
- schneller Einstieg

Nachteile
- limitierte Designmöglichkeiten
- wenig Animationen
- keine interaktive Wireframes

Adobe Experience Design CC

Dieses Prototyping-Tool der „Adobe Creative Cloud“-Familie ist seit Oktober 2017 auf dem Markt, war aber vor dieser Zeit ein Jahr lang in der Beta-Version zu benutzen. Für Anwender, die bereits Erfahrung mit anderen Adobe-Produkten haben, ist

[1] https://www.axure.com
[2] https://www.balsamiq.com
[3] http://www.adobe.com/products/xd.html

der Einstieg und das Erstellen von interaktiven Wireframes mit diesem Tool einfach.

Vorteile

- schnelles Erstellen von komplexen Layouts
- ähnlicher Aufbau wie andere Adobe Produkte
- einfaches Zusammenführen von UI-Elementen

Nachteile

- neu auf dem Markt, daher weniger Dokumentation
- kostenpflichtiges Tool (für Creative Cloud-Abo Inhaber inbegriffen)

Schlussfolgerung

Das Projektteam hat in der Studienarbeit für die Erstellung der Pattern-Icons „Adobe Illustrator CC“ verwendet und ist im Besitze eines Creative Cloud Abos. [4]. Axure und Balsamiq wurden in anderen Studienprojekten eingesetzt. Das Ziel der Wireframes ist, ein Design zu erstellen, mit dem der Betrachter eine gute Übersicht erhält, wie die endgültige Website aussehen könnte. Interaktionsmöglichkeiten geben den Wireframes ein Bild von Lebendigkeit. Aufgrund der fortgeschritten Kenntnisse des Projektteams in „Adobe Illustrator CC“, der einfachen Bedienung des Tools und das Erstellen von dynamischen Wireframes ist die Wahl auf dieses Tool gefallen. Axure ist eine gute Alternative, hat jedoch aufgrund der Komplexität und längeren Einarbeitungszeit den ersten Platz nicht erreicht.

9.2 Unit-Tests

Für die wichtigsten Klassen sind Unit-Tests geschrieben worden. Dies betrifft die Klassen für (i) die Analyse eines eingelesen APIs, (ii) für das Mattern Matching und (iii) Suchfunktion von Patterns in externen URLs.

9.3 Integration-Tests

Jeder Commit in die Versionskontrolle (alle Branches) löst beim Build-Server einen automatisierten Build- und Testprozess aus. Hat ein Test fehlgeschlagen, wurden die Projektmitglieder elektronisch über den Buildfehler informiert. Diese Art von Tests wurden hauptsächlich über den Buildprozess ausgelöst.

[4] https://www.adobe.com/creativecloud.html

Kapitel 10

Ergebnis

In diesem Kapitel wird das Ergebnis der Bachelorarbeit zusammengefasst und bewertet.

Während dieser Arbeit sind zwei verschiedene Endprodukte entstanden. Für neu erarbeitete Patterns der Pattern-Sprache wurden Visualisierungen entworfen. Da die Bewertung der Grafiken sich nicht auf einzelne Icons beschränkt und während der Studienarbeit dokumentiert wurde, werden die Visualisierungen in diesem Kapitel lediglich in zusammengefasster Form reflektiert. Als Endprodukt des Software-Engineering Projektes entstand der Prototyp einer Web-Applikation. Inwiefern der Prototyp die Anforderungen erfüllt ist im Abschnitt „ Bewertung"beschrieben.

10.1 Visualisierungen

Das Feedback der Patternautoren brachte wertvolle Verbesserungsvorschläge mit sich, welche hauptsächlich zu Optimierungen in der visuellen Konsistenz führten. So wurden beispielsweise die beiden Pattern „Atomic Parameter" und „Atomic Parameter List" überarbeitet. Es wurde zurecht bemängelt, dass aus den Grafiken selbst nicht ersichtlich ist, dass sich diese Patterns mit Parametern befassen. Entsprechend wurde die Visualisierungen mit der Bildsemantik der Parameter (schwarzer Punkt) ergänzt. Die Änderungen haben auf die Gesamtbewertung anhand der Qualitätsmerkmale, wie sie in der Studienarbeit gemacht wurde, keinen Einfluss. Nichtsdestotrotz führen die optimierten Details zu einem qualitativ besseren Ergebnis als noch während der Studienarbeit.

Die neu entworfenen Icons reihen sich in die bestehende Bewertung der Studienarbeit ein. Zu betonen ist, dass gerade die Visualisierungen der Service Types beim Kriterium der Multi-Channel-Verwendung schlecht abschneiden, da diese schwierig von Hand nachzuzeichnen sind. Dies liegt daran, dass die Visualisierungen das Kernkonzept der Pattern aufzeigen und sich dieses nicht mit einer simplen Symbolik

darstellen lässt. Umso mehr bestehen die Service Types Visualisierung das Kriterium der Pattern-Kategorie, welche nach wie vor nicht für die ganze Pattern-Sprache als erfüllt betrachtet werden kann.

10.2 Prototyp

Der Prototyp im aktuellen Stadium stellt eine funktionierende Web-Applikation für Interessenten der Pattern-Sprache zur Verfügung. Die Applikation kann als Maven Projekt heruntergeladen und kompiliert werden und ist für ein Docker-Deployement oder für die Verwendung in der Cloud ausgelegt. Für eben erwähnte Deploymentoptionen sind jedoch weitere Entwicklungsschritte nötig, um diese sinnvoll zu nutzen. So hat sich beispielsweise das Projektteam nicht mit den Details der Multiuser-Funktionalitäten auseinander gesetzt. Mehr dazu im Kapitel Inhalte für Folgearbeiten.

Die Stärke der Applikation liegt im Erkennen von Patterns in APIs, die in der OpenAPI Spezifikation beschrieben sind. Mithilfe eines Parsers wird die Struktur automatisch erkannt, was eine genauere Aussage über die Verwendung der Patterns ermöglicht. Die Freitextsuche verfügt über viel Verbesserungspotential, da es im Rahmen diese Arbeit nicht möglich war, eine intelligente Lösung mit maschinellem Lernen umzusetzen. Der Erkennungsalgorithmus basiert auf Keyword-Matching, das aufgrund der unterschiedlichen Beschreibungen von APIs leider noch zu einigen False-Positives führen kann.

Das Frontend entspricht dem Status, den man von eine Prototypen erwarten darf. Die Benutzerinteraktion funktioniert und ist stabil. Der Benutzer wird über wichtige Ereignisse benachrichtigt. Insgesamt gibt es auch in der User Experience Verbesserungspotential.

10.2.1 Bewertung

Die Applikation wird in diesem Abschnitt anhand der Anforderungen bewertet.

Funktionale Anforderungen

In der Planungsphase des Prototyps wurden Funktionale Anforderungen in Form von User Stories festgehalten. Tabelle 10.2.1 zeigt eine Übersicht, ob und wie diese User Stories durch den Prototyp abgedeckt wurden.

Kriterium	Bewertung	Erläuterung
Story 1: API Dokumentation anhand der Icons	Erfüllt	Mit der Applikation ist es möglich, ein API von Grund auf aufzubauen. Der Aufbau eines API ist in vier Abstrakte-Ebenen gegliedert, welche auf unterschiedliche Plattformen übertragen werden können. Ein API kann mit dem API Paver beliebig verändert oder erweitert werden. Die Zuordnung von Patterns auf verschiedenen Ebene wird lediglich durch die Constraints eingegrenzt, welche dazu dienen, dass Patterns nicht falsch angewendet werden.
Story 2: Analyse eines bestehenden APIs im OAS-Format	Erfüllt	OAS-Spezifikationen werden vom Prototypen erkannt und geparsed. Das Format der Spezifikation wird auf die API-Struktur, wie sie in der Applikation verwendet wird, übertragen.
Story 3: Analyse eines bestehenden APIs mit Freitextbeschreibung	Teilweise erfüllt	Die Struktur eines APIs kann nicht aus einer Freitextbeschreibung entnommen werden. Hierfür existieren zu viele Freiheitsgrade in der Gestaltung der API-Beschreibung. Es besteht jedoch die Möglichkeit, in Freitextbeschreibungen nach dem Vorkommen von Pattern zu suchen. In der API Struktur des Prototyps können Referenzen zu Beschreibungen hinterlegt werden, wodurch die Applikation in den Beschreibungen gefundene Patterns dem API auf der entsprechenden Ebene zuordnet.

Kriterium	Bewertung	Erläuterung
Story 4: Editieren des Analyseergebnis	Erfüllt	Das Ergebnis der Analyse wird direkt als Visualisierung dargestellt. Diese Visualisierung kann beliebig editiert werden. Sowohl Felder wie auch Zuordnungen von Patterns können hinzugefügt, angepasst und gelöscht werden.
Story 5: Pattern-Informationen	Erfüllt	Sowohl die Patterns in der Visualisierung wie auch die Patterns im Patternbrowser sind mit der Website der Pattern-Sprache verknüpft. Sämtliche Informationen welche zum Pattern publiziert wurden können so direkt durch einen klick auf das Pattern abgerufen werden.
Story 6: Navigation durch die API Struktur	Erfüllt	Die generierte Visualisierung des API Pavers ist interaktiv. So kann durch Doppelklich auf tiefere Ebenen in eine Komponente hineingezoomt werden. Über die Breadcrumb Navigation oder den API Explorer gelangt man zu den höher gelegenen Ebenen zurück.
Story 7: Schnelle Erfassung grundlegender Eigenschaften	Teilweise erfüllt	Die Kriterien dieser User Story sind stark von subjektiven Ansichten abhängig und schwierig zu messen. Ob die wichtigsten Eigenschaften durch die Pattern-Sprache beschrieben werden und die Patterns im API gefunden werden, konnte das Projektteam im Rahmen diese Projektes nicht ausführlich genug testen und somit die Kriterien auch nicht als vollständig erfüllt bezeichnen.

Kriterium	Bewertung	Erläuterung
Story 8: Vorschläge für die Verwendung weiterer Patterns	Nicht erfüllt	Diese User Story wird durch den Prototypen nicht abgedeckt. Zu Projektstart war der Abgleich von gefundenen und existierenden Pattern nach der Analyse in der Appliaktion geplant, somit hätten Vorschläge für die Verwendung weitere Patterns gemacht werden können. Die Umsetzung war dem Projektteam aufgrund von Zeitmangel und Priorisierung anderer Arbeiten nicht möglich.
Story 9: Einbinden der Visualisierungen in die API-Dokumentation	Erfüllt	Die Darstellungen aus dem API Paver können als HTML-Element exportiert werden und so in eine Website eingebunden werden.

Tabelle 10.2.1: Bewertung Funktionale Anforderungen

Nicht-Funktionale Anforderungen

Tabelle 10.2.2 stellt die Anforderungen vor dem Projektstart dem Endresultat gegenüber.

Kriterium	Bewertung	Erläuterung
Funktionalität: Güte der Analyseergebnisse	Teilweise erfüllt	Diese Kriterium ist für das Pattern Finding im Open API Specification-Format erfüllt. In der Freitextsuche ist das Pattern Finding ist so präzise, wie es die Umsetzung mit einem einfachen Keyword-Matching ermöglicht. Da gewisse Keywords in unterschiedlichen Beschreibungen anders verwendet werden, kann es aufgrund des optimistischen Verfahren zu irrelevanten Ergebnissen kommen.
Funktionalität: Richtigkeit	Erfüllt	Die Patterns sind in der Applikation abgespeichert und werden standardmässig richtig dargestellt.
Benutzbarkeit: Verständlichkeit	Erfüllt	Das Applikation kommt aufgrund der Überschriften und der Gestaltung der einzelnen Komponenten ohne Erklärungstext aus. Aufgrund der Usability Test und den darauf basierenden Verbesserungen findet sich der Anwender in der Applikation zurecht und findet Informationen dort, wo er sie erwartet.
Benutzbarkeit: Erlernbarkeit und Bedienbarkeit	Erfüllt	Die Benutzerinteraktion der Applikation ist einfach konzipiert. Die Bedienelemente welchem dem Benutzer zur Verfügung stehen selbsterklärend. Einzig die Drag and Drop-Funktionalität ist nicht auf den ersten Blick ersichtlich, wird aber in der kurz gehaltenen ”Gettin-StartedAnleitung beschrieben.

Kriterium	Bewertung	Erläuterung
Benutzbarkeit: Feedback	Erfüllt	Fehler aufgrund der falsche Benutzung werden dem Anwender verständlich gemeldet. Technische Fehler werden ebenfalls über den JHipster Errorhandler ausgegeben.
Zuverlässigkeit	Erfüllt	JHipster stellt eine ExceptionTranslater Klasse zur Verfügung, welche das Exception Handling in den Controllern übernimmt (Fehler die von einem Request heraus folgen). Diese Klasse wurde leicht ergänzt. Wo als nötig erachtet, werden Fehler konsequent abgefangen.
Effizienz: Zeitverhalten	Teilweise erfüllt	Die Vorgegebenen Antwortzeiten wurden in den bisherigen Test eingehalten. Da die Geschwindigkeit abhängig von der Rechenleistung und der Grösse des jeweiligen APIs ist, kann hier keine effektive Aussage gemacht werden.
Effizienz: Verbrauchsverhalten	Erfüllt	Die Applikation läuft auf der vorgegebenen Plattform.
Übertragbarkeit	Erfüllt	Das Projekt kann nach dem git clone mit vier Befehl ausgeführt werden, sofern MySQL, Yarn und Maven installiert sind. Docker wird durch JHipster unterstützt und bietet eine weitere Möglichkeit, welche die Übertragbarkeit optimiert.
Randbedingungen	Erfüllt	Die Randbedingungen wurden gemäss der NFR erfüllt.
Lizenzen	Erfüllt	Die verwendeten Technologien entsprechen den Lizenzvorgaben der Anforderungen.

Tabelle 10.2.2: Bewertung nicht-funktionale Anforderungen

10.3 Proof of Concept

Der Dokumentationsassistent bietet im derzeitigen Zustand die Möglichkeit, die vom Projektteam verfasste Schnittstellen-Beschreibung (siehe Absatz 7.3) zu analysieren und als Resultat diese Schnittstellen mit den zugeordneten Patterns anzuzeigen. Die Abbildung 10.3.1 zeigt die im API Browser eingelesene Schnittstellen-Beschreibung in hierarchischer Darstellung. Nebenan ist die Visualisierung des grün hervorgehobenen Calls `apis/id/endpoints` zu sehen.

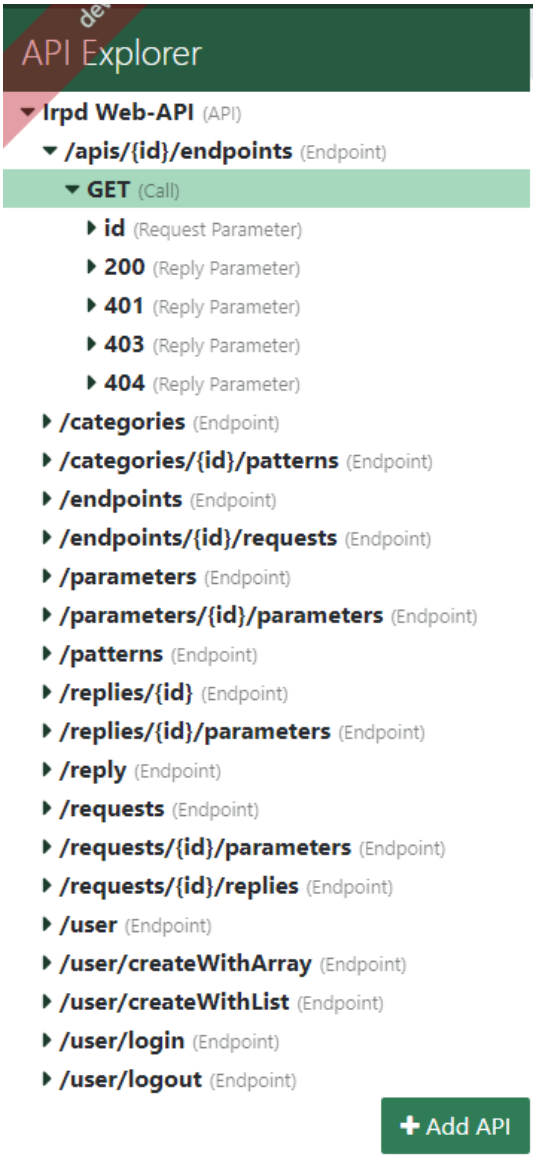


Abbildung 10.3.1: Web API Paver - API Ebene Visualisierung

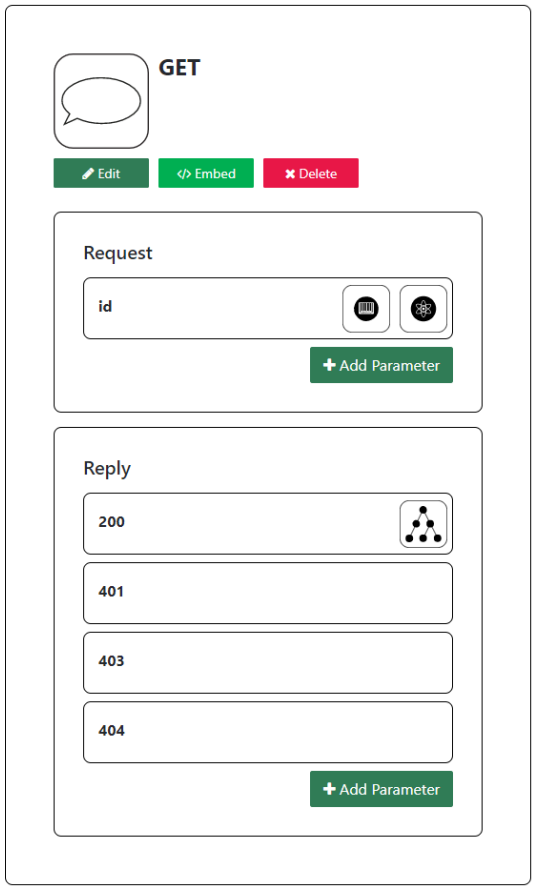


Abbildung 10.3.2: Web API Paver - API Ebene Visualisierung

Aus dieser visuellen Übersicht kann abgeleitet werden, dass es sich um eine GET-Anfrage mit einem Atomic Parameter im Request handelt. Bei einem erfolgreichen Request (200 OK), werden die angeforderten Endpoints in einem Array an den Client retourniert. Diese Struktur entspricht dem „Parameter Tree“ Pattern.

Fazit:
Kennt der Anwender die Patterns und deren Anwendungsmöglichkeiten, so liefert diese visuelle Darstellung eines API eine schnelle Übersicht. Der API-Designer kann verschiedene Lösungen von Grafiken in kurzer Zeit erstellen und diese für weiter Diskussionspunkte benutzen.

Kapitel 11

Inhalte für Folgearbeiten

In diesem Kapitel sind Vorschläge für die Erweiterung und Optimierungen des Prototyps aufgeführt, um diesen Produktionsfähig zu machen.

11.1 Multiuser Betrieb

Um sich als neuen User in der Applikation zu registrieren zu können, muss ein SMTP Server eingerichtet werden. Die Sicherheit der Applikation in Zusammenhang mit den Benutzerberechtigungen war keine Anforderung für die Entwicklung des Prototyps. Requests werden zwar sowohl auf Client- wie auch auf Server-Seite überprüft, jedoch erfordert der Betrieb der Applikation im Internet weitere Massnahmen welche im Rahmen dieses Projektes nicht ergriffen wurden.

11.2 Request Bundling

Während der Entwicklung hat sich aus Zeitmangel an verschiedenen Orten technical Debt angesammelt. Ein wesentlicher Punkt, welcher sich unter anderem beim Skalieren stark auf die Performance auswirken kann, sind die vielen Requests vom Front- zum Backend. Die Patternsprache bietet bereits die Lösung zu diesem Problem an. Einzelne Requests sollen zu einem Bundle zusammengefasst werden, wie es im Pattern „Request Bundle”beschrieben wird.

11.3 Erweiterung Pattern Finding

Das Schlüsselwort-Matching ist ein erster Versuch um die Patterns in API-Beschreibungen zu finden. Dieses Verfahren hat viel Verbesserungspotential, durch den Einsatz von weiteren Technologien und Machine Learning. So könnten zukünftig die Analyseergebnisse detaillierter und weniger fehlerbehaftet ausfallen.

Kapitel 12

Schlussfolgerungen

Der Prototyp des Dokumentationsassistenten entspricht den Anforderungen der Zielgruppen. Mit dem Prototyp ist es für einen API Designer möglich, ein API grafisch zu gestalten. Die textuelle Beschreibung steht nicht mehr im Vordergrund. Mit ein wenigen Klicks kann ein Call gestaltet werden. Damit die Grafik für weitere Verwendungen und Diskussionen von Nutzen ist, sollte den Betrachtern solcher Grafiken die Patterns bekannt sein. Der Prototyp richtet sich auch an Interessierte der Pattern-Sprache, die einen ersten Eindruck in komprimierter Form gewinnen möchten. Das Endprodukt schliesst neue Zielgruppen mit ein.

Der Dokumentationsassistent hat seinen ersten Zweck für das Projektteam bereits erfüllt. Das API für die Web-Applikation wurde erfolgreich eingelesen und analysiert und kann im weiteren Verlauf nach Wunsch visuell erweitert werden. Da es sich um den ersten Prototypen des Web API Paver Dokumentationsassistenten handelt, hat das Projektteam bei der Anwendung des Endproduktes bereits Gedanken zu weiteren Anforderungen an das System notiert. Wird ein API von Grund auf neu aufgebaut, so muss der Benutzer bei der Erfassung eines Calls (mit Request und Reply) repetitive Arbeiten ausführen. Durch die bescheidene Unterstützung durch automatisierte API-Strukturerkennung eignet sich die Applikation im aktuellen Stadium nach Einschätzungen des Projektteams nicht für den produktiven Gebrauch bei grossen Web APIs.

.

Abkürzungen

API Application Programming Interface. 31

BSD Berkeley Software Distribution. 12, 18

CRUD Create Read Update Delete. 42, 46

DOM Document Object Model. 43, 88, *Glossar:* Document Object Model

GPL General Public License. 18

IRP Interface Representation Patterns. 1, 12

ISO International Organization for Standardization. 17

JDL JHipster Domain Language. 31, 88, *Glossar:* JHipster Domain Language

JPA Java Persistence API. 31, 88, *Glossar:* Java Persistence API

JSON JavaScript Object Notation. 31, 46

LGPL Lesser General Public License. 18

MIT Massachusetts Institute of Technology. 12, 18

MVC Model-View-Controller. 31

REST Representational State Transfer. 21, 88, *Glossar:* Representational State Transfer

UI User Interface. 44

WADE Web API Design and Evolution. 5, 88, *Glossar:* Web API Design and Evolution

WSDL Web Service Description Language. 21, 88, *Glossar:* Web Service Description Language

Glossar

AngularJS JavaScript-Framework für den Frontend von Web-Applikationen.. 25

Bootstrap HTML/CSS/JavaScript-Framework mit Vorlagen zur Gestaltung von HTML Seiten. 25

C4-Architekturmodell Software-ArchitekturModel für die visuelle Dartellung des Software Systems. 23

Document Object Model Baustruktur eines HTML Elements. 88

Java Persistence API Das Java Persistence API ermöglicht das Mapping von POJOs (Plain Old Java Obects) auf eine relationale Datenbank mittels Persistenzmodell. 88

JHipster Platform zur Generierung von Webapplikationen basierend auf Angular und Spring. 25, 29, 31

JHipster Domain Language JHipster spezifische Domänensprache. 88

Rate Limit In Web-APIs findet der Rate Limit grosse Anwendung und limitiert den Zugriff auf eine Ressource. 59

Representational State Transfer Architektur- bzw. Integrationsstil für verteilte Systeme, insbesonder Webservices. 88

Spring Boot Vorkonfigurierte Spring Applikation. 25

Spring MVC Framework für Webandwendungen für die Java-Plattform. 25

Swagger Editor Ein Open Source Editor für den Design, die Beschreibung und Dokumentaiton eines API (Swagger-spezifische APIs). 4

Swagger UI UI Komponente für die Visualisierung eines API sowie Interaktion mit einem API (Swagger-spezifische APIs). 4

Web API Design and Evolution Bezeichnung der Domäne in welcher die Patternsprache entwickelt wird. 88

Web Service Description Language Plattformunabhängige Sprache mit welcher messagebasierte Webservices beschrieben werden. 88

Wireframes Drahtgerüst für den konzeptionellen Entwurf einer Website. 70

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] The OAI Announces the OpenAPI Specification 3.0.0. URL https://www.openapis.org/blog/2017/07/26/the-oai-announces-the-openapi-specification-3-0-0. [Zugegriffen am 26.11.2017].

[2] Openapi-specification 2.0. URL https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md. [Zugegriffen am 13.12.2017].

[3] The C4 model for software architecture, 2017. URL https://c4model.com/. [Zugegriffen am 17.12.2017].

[4] Jhipster - Generate your Spring Boot + Angular apps!, 2017. [Zugegriffen am 15.12.2017.

[5] First OAS 3.0 Support in Swagger Core Java Libraries, August 2017. URL https://swagger.io/blog/oas-3-0-support-in-swagger-core-java-libraries/. [Zugegriffen am 13.12.2017].

[6] Overview Swagger-Parser Releases, November 2017. URL https://github.com/swagger-api/swagger-parser/releases. [Zugegriffen am 13.12.2017].

[7] Zeix AG Dr. Sibylle Peuker. Vorlesungsunterlagen zu Human-Computer Interaction Desgin, 2017.

[8] Craig Larman. Uml 2 und Patterns angewendet - Objektorientierte Softwareentwicklung. mitp, 2005.

[9] Josh Long and Kenny Bastani. *Cloud Native Java - Designing Resilient Systems with Spring Boot, Spring Cloud, and Cloud Foundry*. O'Reilly, first edition, August 2017. ISBN 978-1-4493-7464-8.

[10] Matt Raible. *The JHipster mini-book*. Lulu. com, 2016.

[11] Dr. Gernot Starke. *Effektive Software-Architekturen - ein praktischer Leitfaden*. Carl Hanser Verlag, 8 edition, November 2017. ISBN 978-3-446-45207-7.

[12] Sebnem Kaslack und Nicolas Dipner. Studienarbeit - Visualisierung und Umsetzung von Web-API Design Patterns, 2017.

[13] Olaf Zimmermann, Mirko Stocker, Daniel Lübke, and Uwe Zdun. Interface representation patterns-crafting and consuming message-based remote apis. 2017.

[14] Prof. Dr. Zimmermann. Aufgabenstellung Bachelorarbeit - Visualisierung von Interface Representation Patterns: Automatisierung und Werkzeugintegration, 2017.

Anhang A

Schnittstellenbeschreibung Pattern Appearances

Der Vollständigkeit halber sind nachfolgend die Pattern Appearance Schnittstellen in grafischer Form aufgelistet.

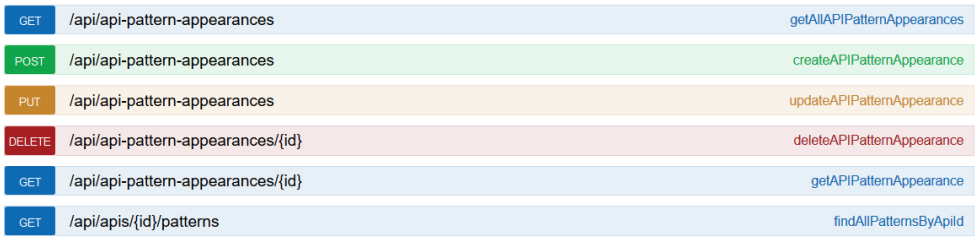


Abbildung A.0.1: API Pattern Appearance Schnittstellen-Übersicht

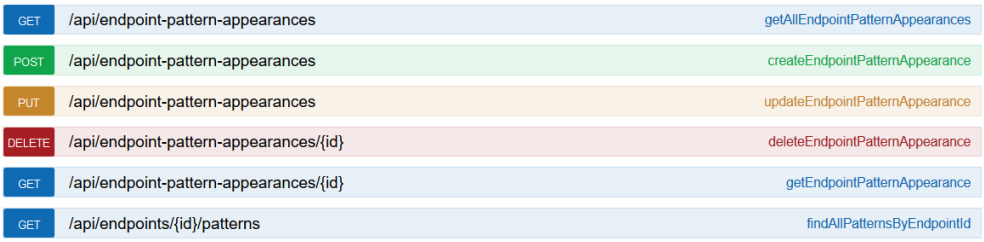


Abbildung A.0.2: Endpoint Pattern Appearance Schnittstellen-Übersicht

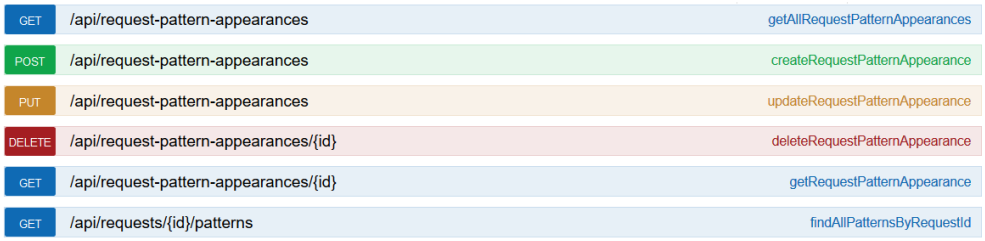


Abbildung A.0.3: Call Pattern Appearance Schnittstellen-Übersicht

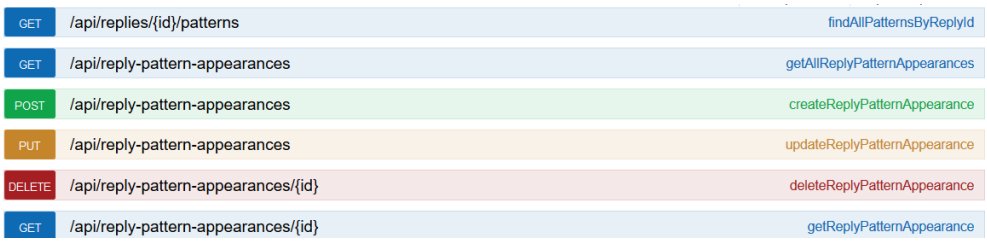


Abbildung A.0.4: Reply Pattern Appearance Schnittstellen-Übersicht

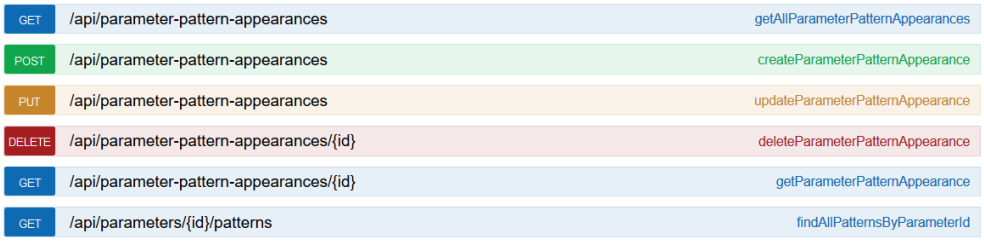


Abbildung A.0.5: Parameter Pattern Appearance Schnittstellen-Übersicht

Anhang B

Pattern Constraints

Nachfolgende Auflistung gibt Auskunft über die Einschränkungen der Patternzuordnungen. Nicht jedes Pattern findet in allen Stufen (API, Endpoint, Call, Reply und Parameter) Anwendung. Diese Einschränkungen können in der Web-Applikation angepasst werden. Es muss darauf geachtet werden, dass diese kommasepariert erfasst werden.

Nr	Patternname	Constraints
1	Atomic Parameter	Parameter,NestedParameter,Request,Reply
2	Atomic Parameter List	Parameter,NestedParameter,Request,Reply
3	Parameter Tree	Parameter,NestedParameter,Request,Reply
4	Parameter Forest	Parameter,NestedParameter,Request,Reply
5	Annotated Parameter Collection	Parameter,NestedParameter,Request,Reply
6	Cursor-Based Pagination	Parameter,NestedParameter,Request,Reply
7	Offset-Based Pagination	Parameter,NestedParameter,Request,Reply
8	Time-Based Pagination	Parameter,NestedParameter,Request,Reply
9	Public API	API
10	Community API	API
11	Solution-Internal API	API
12	Vertical Integration	API
13	Horizontal Integration	API
14	Service Contract	API
15	Master Data Resource	Endpoint,Parameter,NestedParameter,Request,Reply
16	Transactional Data Resource	Endpoint,Parameter,NestedParameter,Request,Reply
17	Embedded Reference Data	Endpoint,Parameter,NestedParameter,Request,Reply
18	Linked Reference Data	Endpoint,Parameter,NestedParameter,Request,Reply
19	Reference Data Lookup	Endpoint,Parameter,NestedParameter,Request,Reply

Nr	Patternname	Constraints
20	Id Parameter	Parameter,NestedParameter
21	Entity Parameter	Parameter,NestedParameter
22	Link Parameter	Parameter,NestedParameter
23	Provenance Metadata Parameter	Parameter,NestedParameter
24	Control Metadata Parameter	Parameter,NestedParameter
25	Aggregated Metadata Parameter	Parameter,NestedParameter
26	Command Service	Endpoint,Request
27	Information Service	Endpoint,Request
28	Query Services	Endpoint,Request
29	Validation Service	Endpoint,Request
30	Service Level Agreement	API,Endpoint,Request
31	Metering and Billing	API,Endpoint,Request
32	API Key	API,Endpoint,Request,Reply,Parameter,NestedParam.
33	Rate Limit	API,Endpoint,Request,Reply
34	Wish List	Request
35	Wish Template	Request
37	Conditional Request	Request
38	Request Bundle	Request
39	Context Representation	Request,Reply
40	Error Reporting	Reply
41	Version Identifier	API,Endpoint
42	Semantic Versioning	API,Endpoint
43	Eternal Lifetime Guarantee	API,Endpoint
44	Limited Lifetime Guarantee	API,Endpoint
45	Two in Production	API,Endpoint
46	Aggressive Deprecation	API,Endpoint
47	Experimental Preview	API,Endpoint
48	Parameter List	Parameter,NestedParameter,Request,Reply
49	Parameter	Parameter,NestedParameter,Request,Reply

Tabelle B.0.1: Übersicht Pattern Constraints

Anhang C

Anpassungen von Pattern- und Kategorie-Bezeichnungen

Es ist davon auszugehen, dass sich Pattern -und Kategorienamen nicht all zu oft ändern werden. Tritt der Fall ein, dass für ein Pattern oder eine Kategorie eine Namensänderung durchzuführen ist, so können die folgende Schritte durchlaufen werden.

1. Unter src\main\resources\config\die Datei *application-patternconstants.properties* mit einem beliebigen Editor öffnen.
2. Namensanpassung wunschgemäss vornehmen und speichern.
3. Applikation/Server neu starten.

Hinweis: Im Konfigurationsfile sind nur jene Patterns aufgelistet, welche für das Pattern-Matching der Einlese- und Analysefunktion relevant sind. Für das Anzeigen des Constraints auf der Benutzeroberfläche, ist es wichtig, auch den entsprechenden Kategorienamen bei Änderung anzupassen. Die Abbildung C.0.1 zeigt den Inhalt des Konfigurationsfiles.

```
#Pattern matching properties
PATTERN_SEMANTIC_VERSIONING= Semantic Versioning
PATTERN_SERVICE_LEVEL_AGREEMENT= Service Level Agreement
PATTERN_API_Key= Api Key
PATTERN_VERSION_IDENTIFIER= Version Identifier
PATTERN_ATOMIC_PARAMETER_LIST= Atomic Parameter List
PATTERN_REQUEST_BUNDLE= Request Bundle
PATTERN_RATE_LIMIT= Rate Limit
PATTERN_ID_PARAMETER= Id Parameter
PATTERN_ATOMIC_PARAMETER= Atomic Parameter
PATTERN_PARAMETER_FOREST= Parameter Forest
PATTERN_PARAMETER_TREE= Parameter Tree
PATTERN_ERROR_REPORTING = Error Reporting
PATTERN_CATEGORY_STRUCTURAL_REPRESENTATION = Representation Patterns
```

Abbildung C.0.1: Auszug aus application-patternconstants.properties