|  |
| --- |
| Universität Potsdam |
| Tool.UP |
| Dokumentation |

|  |
| --- |
| Thiemo Belmega  15.06.2016 |

Inhalt

[Einleitung 3](#_Toc453852615)

[Userstories 3](#_Toc453852616)

[Userstory 1 3](#_Toc453852617)

[Userstory 2 4](#_Toc453852618)

[Userstory 3 4](#_Toc453852619)

[Architektur 5](#_Toc453852620)

[Technologien und Bibliotheken 6](#_Toc453852621)

[Datenmodell und Datenbankschema 6](#_Toc453852622)

[Backend 7](#_Toc453852623)

[de.uni-potsdam.cs.toolup.model 7](#_Toc453852624)

[de.uni-potsdam.cs.toolup.database 7](#_Toc453852625)

[Rest-API 8](#_Toc453852626)

[Tests 8](#_Toc453852627)

[Front-End 8](#_Toc453852628)

GitHub:

<https://github.com/tbelmega/Tool.UP>

<https://github.com/tbelmega/Tool.UP-frontend-backbone>

Kontakt:

t.belmega@gmx.de

# Einleitung

Der Webservice Tool.UP präsentiert strukturierte und durchsuchbare Informationen über E-Learningwerkzeuge und ihre Funktionalität.

Der Nutzer navigiert zu den gewünschten Informationen über das Web-Frontend, das Abfragen zur REST-Schnittstelle des Backends schickt. Die Daten werden vom Betreiber von Tool.UP in einer SQL-Datenbank gepflegt, von wo das Backend die Daten abfragt und weiterverarbeitet.

Die vorliegende Dokumentation beschreibt die einzelnen Komponenten von Tool.UP und ihren groben Aufbau. Darüber hinaus folgt Tool.UP dem Ideal von *selbstdokumentierendem Code*: Durch beschreibende Variablen- und Methodennamen sowie kurze Methoden und Klassen mit klaren Verantwortlichkeiten ist der Code so gehalten, dass er möglichst lesbar und verständlich ist. Die mitgelieferten Unit-Tests und Integrationstests dienen gleichzeitig als Code-Beispiele, die dem Entwickler zeigen, wie die Klassen und Methoden des Produktiv-Codes zu verwenden sind. Eine vollständige Dokumentation des Codes durch Diagramme oder JavaDoc wird vermieden, da diese Art Dokumentation schnell veraltet und fehleranfällig wird, wodurch sie mehr Schaden als Nutzen verursacht (vgl. Robert C. Martin 2008, „Clean Code“).

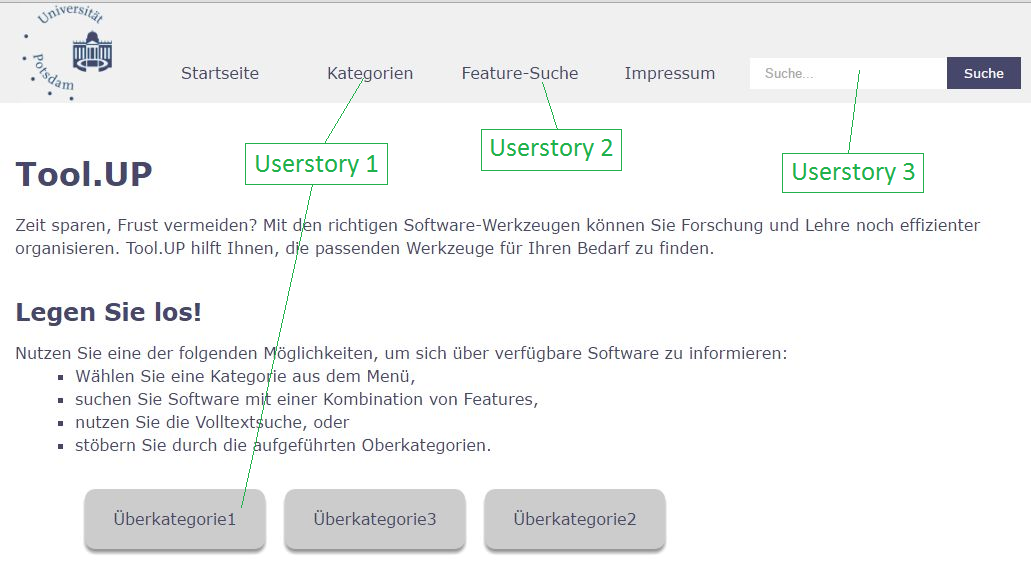
# Userstories

Tool.UP deckt folgende Userstories ab:

## Userstory 1

Absicht: Der Nutzer möchte sich einen Eindruck von verfügbaren E-Learning-Werkzeugen machen.

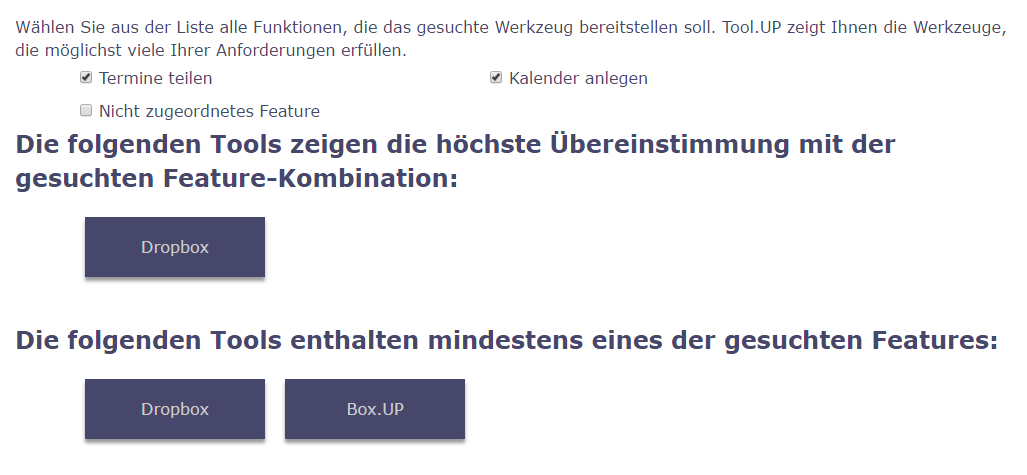
Umsetzung: Der Nutzer beginnt mit einer Liste von Kategorien und navigiert durch den Baum von Ober- und Unterkategorien und den dazugehörigen Applikationen. Zum Einstieg nutzt er entweder auf der Startseite alle Kategorien ohne weitere Oberkategorie, oder auf der Kategorien-Seite alle Kategorien mit direkt zugeordneten Applikationen.



## Userstory 2

Absicht: Der Nutzer sucht nach einem E-Learning-Werkzeug, das bestimmte Funktionen erfüllt.

Umsetzung: In der Feature-Suche legt der Nutzer eine Kombination von Features fest, die seinen Wünschen entspricht. Tool.UP zeigt ihm auf derselben Seite die besten Suchergebnisse sowie alle Applikationen mit mindestens einem der gesuchten Features.



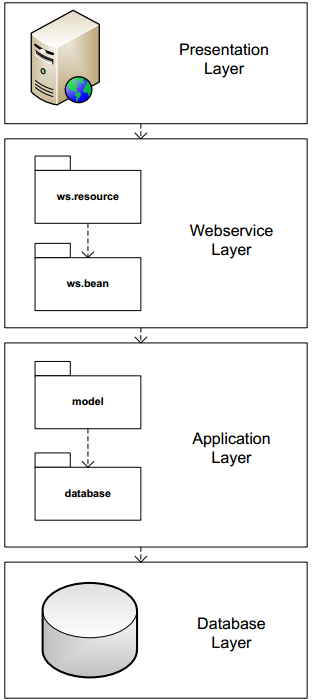
## Userstory 3

Absicht: Der Nutzer möchte eine Applikation finden, die er dem Namen nach kennt oder die er bei einer früheren Nutzung von Tool.UP gesehen hat.

Umsetzung: Der Nutzer gibt ein oder mehrere Worte in die Textsuche ein. Er erhält eine Liste aller Applikationen, deren Name, Beschreibung oder Features zu der Suchanfrage passen.

# Architektur

Tool.UP ist in vier Schichten organisiert. Kennzeichen einer Schichten-Architektur ist, dass die Komponenten jeder Schicht nur aufeinander und auf die direkt darunterliegende Schicht zugreifen. Keine Komponente greift auf eine höher- oder noch tieferliegende Schicht zu. Diese Architektur stellt eine möglichst lose Kopplung der Komponenten sicher, sodass man Teile des Systems verändern oder austauschen kann, ohne das gesamte System anpassen zu müssen.

Die **Präsentationsschicht** umfasst das Web-Fontend von Tool.UP, aufgebaut als Single-Page-Application mit dem Framework backbone.js. Sie umfasst eine HTML-Seite, ein CSS, und jeweils einer JavaScript-Datei für jede Ansicht (Backbone View) und jedes Datenmodell (Backbone Models und Collection). Backbone lädt die Daten für jedes benötigte Model über eine Http-Anfrage ans Backend.

Im Backend wird die Http-Anfrage vom REST-Framework Jersey entgegengenommen und als Methodenaufruf einer Webressource ausgeführt (**Webservice-Schicht**).

Webressourcen sind annotierte Methoden im Paket *resource*. Jersey erwartet als Antwort Java-Objekte im JavaBean-Format, die es dann in JSON[[1]](#footnote-1) umwandelt und in eine Http-Antwort packt.

Die JavaBeans sind im Paket *bean* definiert. Sie dienen als Wrapperklassen für die Domänenobjekte aus der Anwendungsschicht.

Die Anwendungsschicht ist unabhängig vom Einsatz als Webservice. Es ist daher möglich, statt der Webservice-Schicht alternative Benutzerschnittstellen auf die Anwendungsschicht aufzubauen ohne sie zu verändern, zum Beispiel eine lokale grafische Benutzeroberfläche in Swing.

Die **Anwendungsschicht** umfasst die Java-Pakete *database* für den Zugriff auf die Datenbank, und das Paket *model* für die Datenhaltung im Backend und die Anwendungslogik. Tool.UP setzt einen Objektcache ein, um die Anzahl der Datenbankzugriffe zu reduzieren. Dabei werden Objekte nach dem Laden aus der Datenbank im Cache gehalten und bei erneutem Zugriff auf dieselbe ID direkt aus dem Cache verwendet.

Die **Datenbankschicht** besteht aus einer SQL-Datenbank zur Datenhaltung. Über einen externen Client können die Daten hier auch verändert werden. Das Backend von Tool.UP kann zwar bereits Daten in die Datenbank einfügen, updaten und löschen, vorerst wird diese Funktionalität aber nur von den UnitTests genutzt; es gibt in Tool.UP keine Nutzeroberfläche zum Verändern von Daten.

# Technologien und Bibliotheken

Die folgende Tabelle listet die verwendeten Technologien und Bibliotheken von Drittanbietern auf:

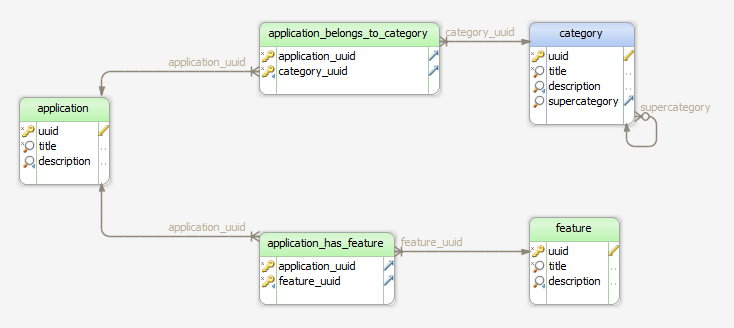
|  |  |
| --- | --- |
| **Verwendungszweck** | **Technologie** |
| Datenbankserver | MySQL Server 5.6 |
| Datenbankkonnektor im Backend | jDBC |
| Backend | Java 8 |
| Backend Test-Framework | TestNG |
| Build-Tool | Apache Maven |
| Backend-Bibliothek | Apache Commons Lang |
| Java-Framework für REST-Schnittstellen | Jersey |
| Verarbeitung von JSON-Daten in Java | org.json |
| Objektcache | EHCache |
| Frontend | HTML, CSS, Javascript |
| Frontend Bibliotheken | jquery, underscore.js, backbone.js |

# Datenmodell und Datenbankschema

Die in Tool.UP betrachteten Entitäten sind E-Learning-Programme und ihre Funktionalität. Außerdem werden die E-Learning-Programme in eine Hierarchie von Kategorien eingeteilt.

Im Datenbankschema werden diese Entitätsklassen repräsentiert durch die Tabellen Application, Feature und Category. Applications und Features stehen in viele-zu-viele-Beziehung (n:m), Applications und Categories ebenso. Dadurch werden zwei Beziehungstabellen notwendig. Außerdem kann jede Category eine Category als Überkategorie besitzen, dadurch steht die Tabelle Category in eins-zu-viele-Beziehung (1:n) zu sich selber.

Das vollständige Datenbankschema ist im Diagramm abgebildet.



**Konvention**: Die ID-Werte der Datensätze müssen mit den Teilstrings „application-“, „category-“ oder „feature-“ beginnen, entsprechend ihrer jeweiligen Tabelle. Dieses Präfix wird vom Java-Backend benutzt, um den Datensatz zu einer gegebenen ID aus einer Tabelle zu laden.

# Backend

Das Backend von Tool.UP ist in Java 8 geschrieben.

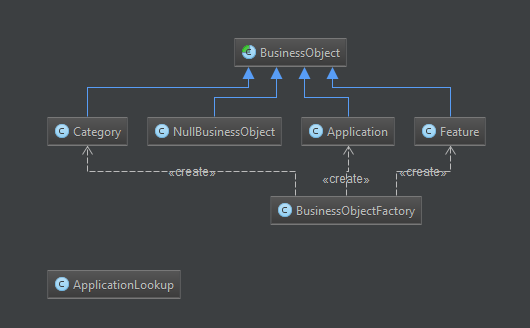
Es kommuniziert mit der Datenbank über den JDBC-Datenbankkonnektor. Dabei werden vorbereiete SQL-Abfragen aus der Datei SQL\_Statements.xml gelesen und ausgeführt.

Mit der Frontend kommuniziert das Backend über eine REST-Schnitstelle, die mit Hilfe der javax.rs-Referenzimplementierung Jersey gestaltet ist.

Der Java-Code des Backends besteht aus mehreren Paketen, die im Folgenden beschrieben sind.

## de.uni-potsdam.cs.toolup.model

Das Paket **model** stellt die Klassen für die Domänenobjekte von Tool.UP bereit (Application für E-Learning-Programme, Feature für deren Funktionen und Category für die Kategorien-Hierarchie), sowie die Methoden zu ihrer Verarbeitung.



Die Klassen Application, Category und Feature entsprechen den Enitätsklassen des Datenbankmodells. Ihre gemeinsame Oberklasse BusinessObject enthält die Funktionalität, um eine JSON-Repräsentation der Java-Objekte zu erzeugen. Die BusinessObjectFactory erzeugt Java-Instanzen auf Grundlage der Datenbank-Antworten. Das NullBusinessObject repräsentiert das Nichtvorhandensein eines BusinessObjects, zum Beispiel als Ergebnis eine Datenbankabfrage (vgl. das Entwurfsmuster NullObject in der Literatur).

Die Klasse ApplicationLookup repräsentiert eine Menge von Applications als Ergebnis einer Feaure-Suche. Eine Instanz von ApplicationLookup nimmt eine Liste von Feature-IDs entgegen und identifiziert alle Applications aus der Datenbank, die eine größtmögliche Menge der gegebenen Features abdecken. Im besten Fall liefert der ApplicationLookup also alle Applications, die alle gewünschten Features beinhalten; wenn es keine solche Applications gibt, liefert er alle Applications, die höchstens eines der gesuchten Features nicht aufweisen, und so weiter.

## de.uni-potsdam.cs.toolup.database

Das Paket **database** ist für die Kommunikation mit der Datenbank verantwortlich.

Die Klasse ToolUpProperties lädt Einstellungen wie Serveradresse, Nutzername und Passwort aus der Datei Tool.UP\_cfg.xml.

Die Klasse SqlStatementFactory erzeugt SQL-Abfragen als JavaObjekte, indem sie den benötigten SQL-Code aus der Datei SQL\_Statements.xml einliest.

Der DatabaseController (Singleton-Entwurfsmuster) bildet den Adapter zur Datenbank für das übrige Backend. Er stellt alle Methoden zur Datenbankabfrage zur Verfügung, holt die PreparedStatement-Objekte dafür aus der SqlStatementFactory und führt sie aus.

# Rest-API

# Tests

# Front-End

1. **J**ava**S**cript **O**bject **N**otation, ein gängiges Datenaustauschformat zwischen Programmiersprachen. Entspricht der Syntax einer Objektdefinition in JavaScript. [↑](#footnote-ref-1)