Realisierungsbericht   
Wissensdatenbank

| Auftraggeber | Georg Ninck |
| --- | --- |
| Projektleiter | Joel Häberli |
| Autor | Joel Häberli |
| Klassifizierung | Nicht klassifiziert |
| Status | Zur Prüfung |
|  |  |

Änderungsverzeichnis

| Datum | Version | Änderung | Autor |
| --- | --- | --- | --- |
| 21.11.2017 | 0.1 | Draft | David Schor |
| 12.11.2017 | 0.2 | Ergänzung | David Schor, Anurally Keller, Joel Häberli |
| 13.12.2017 | 1.0 | Finish | Joel Häberli |

Inhaltsverzeichnis

1 Zusammenfassung 3

2 Technische Detailspezifikation 3

2.1 Systemdesign 3

2.1.1 Struktur 3

2.1.2 Dynamik 4

2.2 Schnittstellendefinitionen 4

2.3 Datenmodell 5

2.4 Sicherheit (ISDS) 5

3 Systemdokumentation 6

3.1 Anwendungshandbuch 6

3.1.1 Übersicht 6

3.1.2 Funktionen und Detailbeschreibung 6

3.1.3 Fehlerbehandlung 8

3.2 Integrations- und Installationshandbuch 8

3.3 Betriebshandbuch 8

4 Testspezifikation 9

4.1 Unit-Test 9

4.2 Systemtest 9

4.2.1 Testfälle 9

5 Testprotokoll 13

5.1 Systemtest 1 13

5.1.1 Testfall 1 „Neues Benutzerkonto erstellen “ 13

5.1.2 Testfall 2 „Anmelden“ 13

5.1.3 Testfall 3 „Page erstellen“ 13

5.1.4 Testfall 4 „Tags hinzufügen“ 13

5.1.5 Testfall 5 „Tags bearbeiten“ 13

5.1.6 Testfall 6 „Page bearbeiten“ 13

5.1.7 Testfall 7 „Page suchen“ 13

5.1.8 Testfall 8 „Page löschen“ 13

5.1.9 Testfall 9 „Page exportieren“ 13

5.1.10 Testfall 10 „Passwort ändern“ 13

5.1.11 Testfall 11 „Benutzer deaktivieren (löschen)“ 13

5.1.12 Testfall 12 „Benutzer reaktivieren“ 13

6 Projektplanung 14

Anhang A: Quellcode 15

6.1 API 15

6.2 Connection 28

6.3 Entity 28

6.3.1 Data 28

6.3.2 Domain 33

6.4 WDB 39

6.5 Resources 39

# Zusammenfassung

Die Struktur der Wissensdatenbank wurde so einfach wie möglich gehalten. Deshalb gibt es auch nur zwei Pakete; API und Entity. Die API übernimmt die Geschäftslogik und exponiert eine Schnittstelle. Diese kommuniziert mit JSON. Mithilfe des Paketes Entity, werden die Daten persistiert. Die Schnittstelle ist eine herkömmliche Spring-REST-API und wird wie jede andere REST-API über das http-Protokoll angesprochen.

Die Sicherheit der Daten wird über ein SessionToken, welches bei jedem API Aufruf den Benutzer identifiziert, sichergestellt. Weil wir mit dem Spring-Framework arbeiten, sind wir automatisch mit den Top-10 der OWASP Richtlinien geschützt.

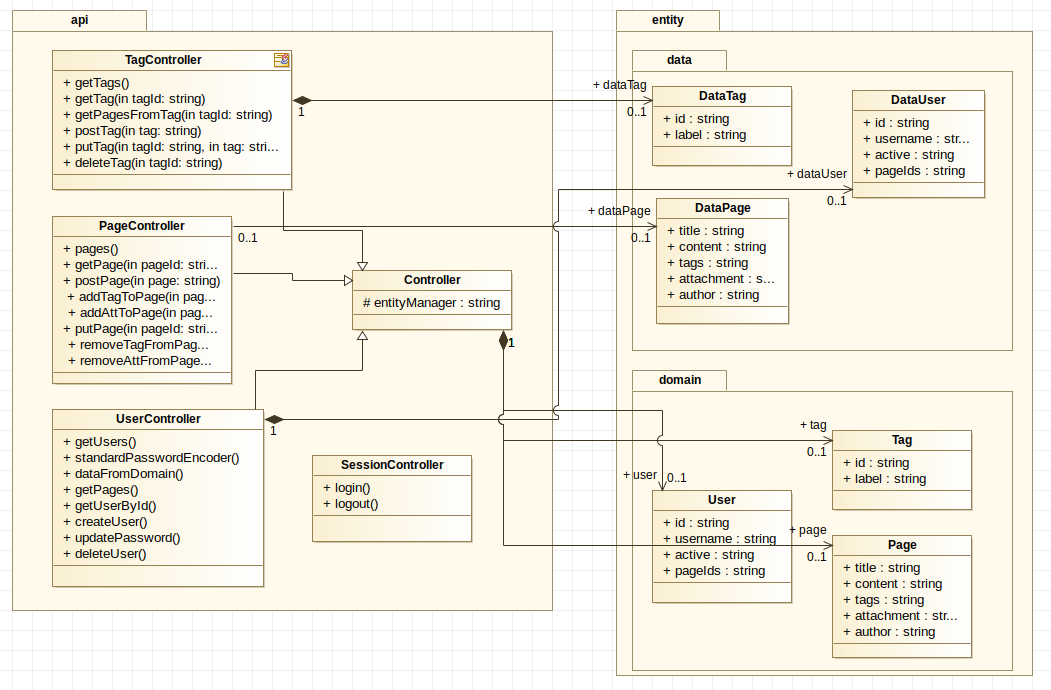
Alle Kriterien des Akzeptanztestes wurden eingehalten.

Für technisch versierte und interessierte ist der Quellcode im Anhang zu finden.

# Technische Detailspezifikation

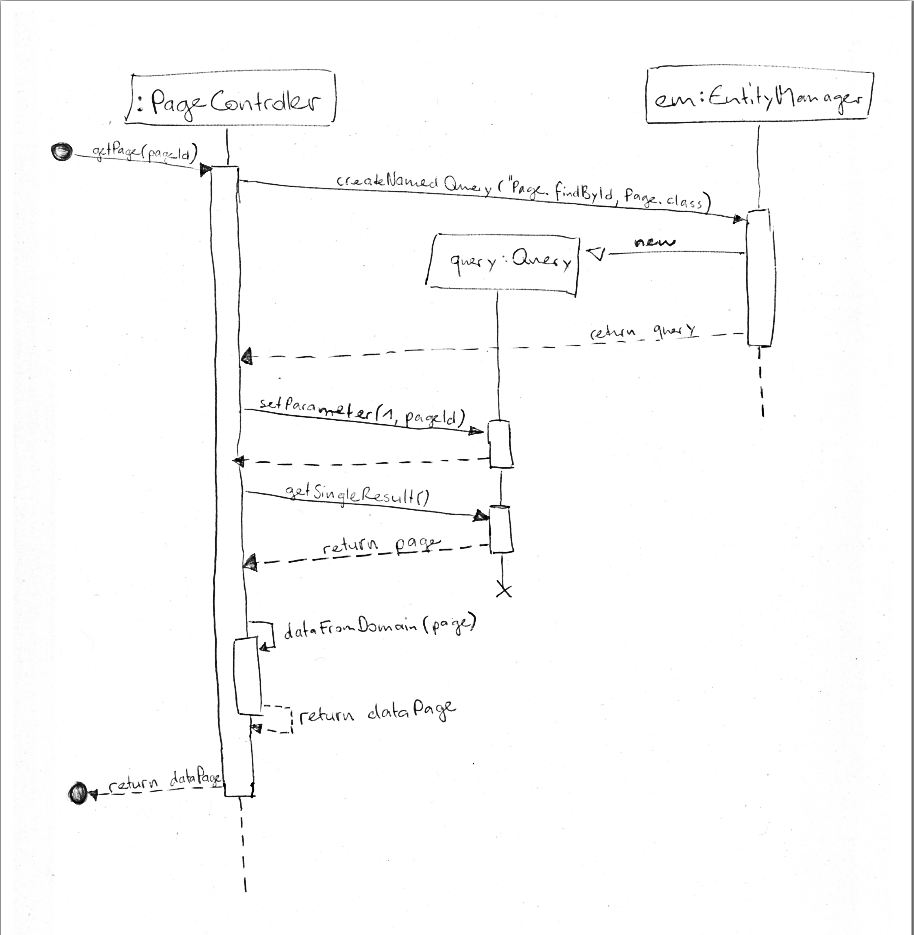
## Systemdesign

### Struktur

**

Unsere Applikation ist wie oben abgebildet in zwei Pakete aufgeteilt. Die API und die Entities. Die Entities sind wiederum in zwei Subpackete aufgeteilt; "data" und "domain". In der API hat es Controller. Die Controller holen Daten aus der Datenbank über die Datenklassen domain-Package. Diese sind Datenklassen und funktionieren mit der JPA. Damit man nicht mit den Objekten direkt auf der Datenbank arbeitet, gibt es die Klassen im Package "data". Mit diesen Datenklassen arbeitet die Applikation. Wenn die Änderungen abgeschlossen sind, werden sie wieder mit den "domain"-Klassen persistiert. Ein wichtiger Controller ist der SessionController. Er erzeugt beim Login ein Session-Token, welches den eingeloggten Benutzer identifiziert.

### Dynamik



Dieses Sequenzdiagramm zeigt den Ablauf der API auf, wenn jemand eine bestimmte Page aufruft. Es startet beim PageController und hört dort auch wieder auf, da Spring den Rest automatisch übernimmt.

## Schnittstellendefinitionen

Unsere Applikation hat keine Schnittstellen, welche sie konsumiert. Unsere Applikation ist jedoch eine API und somit selbst eine Schnittstelle. Diese Schnittstelle wird durch die verschiedenen Controller (UserController, TagController, AttachmentController und PageController) abgebildet.

Unsere Schnittstelle versendet und empfängt JSON-Daten. Man ruft die Schnittstelle über das HTTP-Protokoll auf. Dies, weil unsere Schnittstelle mit den zugehörigen Methoden (GET, PUT, POST, etc.) arbeitet und nur Daten entgegennimmt, welche mit den richtigen Methoden versendet werden. Das erhöht auch die Sicherheit der Applikation.

Unsere Schnittstelle kann von jedem User-Interface benutzt werden, welches Daten im JSON-Format ein- und auslesen kann.

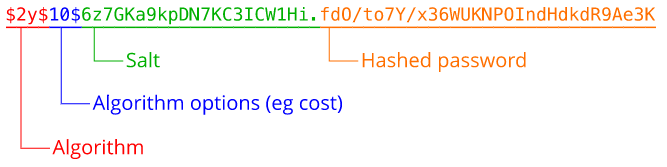
## Datenmodell

Wir arbeiten mit JPA und Hibernate. Deswegen haben wir kein statisch vorgegebenes ERM.

Die betreffenden Klassen sind bereits weiter oben im UML-Klassendiagramm, im Package ‚domain‘ zu sehen. Es sind alles normale JPA-Datenklassen und mit ‚@Entity‘ annotiert. Die Klassen sind selbsterklärend: Der User wird in der Klasse ‚User‘ abgebildet. Eine Page in der Klasse ‚Page‘, deren Tags in der Klasse ‚Tag‘ und die Attachments der Page in der Klasse ‚Attachement‘.

## Sicherheit (ISDS)

Der Inhalt der der Wissensdatenbank kann privat sein. Deshalb müssen die Daten sorgfältig behandelt werden.

Dabei werden keine persönlichen oder besonders schützenswerte Daten gespeichert.  
Der Benutzer muss für seinen Account ein Passwort angegeben. Dieses muss mit SHA256 verschlüsselt werden. Dies kann mithilfe von Salt und einem zusätzlichen Algorithmus verstärkt werden.  


Da wir die API-Variante verwenden, kann der Benutzer weiterhin die Verantwortung auf die einzelnen Daten haben, in dem er die Wissensdatenbank selbst hosted.  
Dennoch muss aus der Sicht der Entwicklung weiterhin die Sicherheit gewährleistet werden.

Bevor irgendwelche Daten in die Datenbank abgespeichert werden, sollten diese auf verdächtige Attacken untersucht werden, wie zum Beispiel eine Injection. Diese kann durch named Statements grösstenteils verhindert werden. Nebenbei wird der Input als String abgespeichert und dann als Object weiterverarbeitet. Dies unterstützt die Sicherheit der Wissensdatenbank. Die Attachements, welcher der Benutzer hinzufügen kann, werden nicht ausgeführt. Dies ist auch bei der Page so. Somit wird kein Code z.B. in einem Bild oder sonstiges ausgeführt.

Die MySQL Datenbank muss sicher aufgesetzt werden, um Fehler zu verhindern, welche Schwachstellen hervorrufen könnten. Dazu gehört auch eine korrekte Konfiguration der Datenbank:

* Password für den **root** Benutzer setzten
* Andere unnötige Accounts löschen oder korrekte Berechtigungen vergeben
* Benutzername für den **root** Benutzer ändern
* Unnötige Services deaktivieren
* Remote Access deaktivieren
* Andere unnötige Datenbanken löschen
* Root Directory ändern
* Server auf dem neusten Stand

# Systemdokumentation

## Anwendungshandbuch

### Übersicht

Der Zweck von der Wissensdatenbank ist: Das Gelernte und die Erfahrungen einfach und schnell speichern zu können und dies als Lehrmittel und Nachschlagewerk wiederzuverwenden. Zusätzlich wird die Übersicht der erstellten Notizen gewährleistet.

### Funktionen und Detailbeschreibung

Die Wissensdatenbank ist eine REST API, welche mit JSON kommuniziert. Folgend werden diese Begriffe kurz erklärt und wie man die API verwendet. Dazu werden die verschiedenen Funktionen aufgelistet.

**Was ist eine REST API**

Die REST API kommt bei der Wissensdatenbank zum Einsatz. Die REST API beschreibt, wie verteilte Systeme miteinander kommunizieren können.   
Sie wird auf dem Server initialisiert und kann durch Befehle, von einem Computer, aufgerufen werden.

Um mehr Details zu erfahren, werden folgend auf einige Webseiten referenziert:

* <http://www.creativeworkline.com/2011/02/rest-und-das-interface-gehoert-dir/>
* <https://www.cloudcomputing-insider.de/was-ist-eine-rest-api-a-611116/>

**Was ist JSON**

Um mit der REST API kommunizieren zu können, verwendet die Wissensdatenbank JSON.   
JSON ist somit kein Protokoll, sondern nur ein Datenformat. Dabei wird es hauptsächlich zwischen Server und Client verwendet. Es werden zum Beispiel, bei einer Anmeldung, der Benutzername und das Passwort abgespeichert. Danach wird die JSON Datei an die REST API gesendet und diese validiert die Daten.

Um mehr Details zu erfahren werden folgend auf einige Webseiten referenziert:

* <https://de.wikipedia.org/wiki/JavaScript_Object_Notation>
* <https://joocom.de/lexikon/was-ist-json/>

Um die verschiedenen Funktionen verwenden zu können, muss ein Request mit JSON erfolgen. Die Funktion muss im Requestbody platziert werden. Danach wird das JSON mit den verschiedenen Befehlen an den Server gesendet.

**Beispiel**

Einen Request auf eine API kann von jeder Programmiersprache durchgeführt werden. Sie haben meist nicht die gleiche Syntax. Somit wird nicht ein explizites Beispiel mit einer Programmiersprache gemacht, sondern ein Beispiel, wie man einen Request an die API senden kann:

Um ein Benutzer zu erstellen, muss zuerst die folgende URL aufgerufen werden:   
*{baseUrl}/user*Die *{baseUrl}* wird mit ihrem Host und ohne geschweifte Klammern ersetzt.

Danach werden folgende Werte im Requestbody benötigt:  
*{ Name, Password }*

Alle Funktionen werden folgend in einer Tabelle aufgeführt. Sie werden gleich implementiert wie der bereits oben erwähnten Funktion.

|  |  |
| --- | --- |
| GET (Lesen)  Alle Get-Aufrufe können mit Paramter verfeinert werden. Beispiel (sucht nach Pages mit «HTML» im Titel): page?title=html  Alle Get-Aufrufe können in «Seiten» aufgeteilt werden. Dazu dienen die Parameter offset und count. Beispiel (liefert Page 50 - 100): page?offset=50&count=50 | |
| tag | Alle Tags |
| tag/id | Tag mit Id |
| tag/id/page | Alle Pages mit Tag |
| page | Alle Pages |
| page/id | Page mit Id |
| attachement/id | Attachement mit Id |
| attachement/id/file | Die Datei |
| attachement/id/page | Page des Attachement |
| user | Alle User |
| user/id | User mit Id |
| user/id/page | Alle Pages des Users |

|  |  |
| --- | --- |
| POST (Erstellen) | |
| tag { Label } | Erstelle Tag |
| page { Title, Content, AuthorId, TagIds[] } | Erstelle Page |
| user { Name, Password } | Erstelle User |
| page/id/tag { TagId } | Tag einer Page hinzufügen |
| page/id/attachement { Attachement } | Attachement einer Page hinzufügen |
| login { Name, Password } | Erstellt eine Session für einen User |
| logout | Löscht die Session |

|  |  |
| --- | --- |
| PUT (Ändern) | |
| tag/id { Label } | Ändert Label des Tags |
| page/id { Title, Content } | Ändert Titel/Content der Page |
| user/id { Old Password, New Password } | Ändert das Passwort |

|  |  |
| --- | --- |
| DELETE (Löschen) | |
| tag/id | Löscht den Tag (wenn keine Page mit diesem Tag existiert) |
| page/id | Löscht die Page |
| user { Password } | Löscht den eigenen Account |
| page/id/tag { TagId } | Entfernt einen Tag von einer Page |
| page/id/attachement { AttachementId } | Entfernt Attachement von einer Page |
| attachement/id | Löscht Attachement |

### Fehlerbehandlung

#### Fehler 503: Service Unavailable

Der Server ist offline oder hat keine Verbindung zum Internet. Melden Sie dies ihrem Systemadministrator oder überprüfen Sie, ob der Server online ist und eine Verbindung zum Internet hat.

#### Keine Verbindung: ERR\_INTERNET\_DISCONNECTED

Dieser Fehler taucht auf, wenn Ihr Client keine Verbindung ins Internet hat. Versuchen Sie sich mit einem funktionierenden Access Point, welcher Zugang zum Internet hat, zu Verbinden. Wenn nötig starten Sie ihren Client neu.

#### Fehler 500: Internal Server Error

Dieser Fehler wird vom Server verursacht. Dabei ist der Fehler meist unterschiedlich und kann mehrere Ursachen haben. Kontaktieren Sie ihren Systemadministrator.

#### Fehler 400: Bad Request

Der Request wurde falsch ausgeführt. Ihre Web-Applikation, welche die API verwendet, verursacht einen Fehler, indem sie den Request falsch sendet oder nicht vollständig ausgefüllt wurde. Überprüfen Sie Ihre Requests an die API.

## Integrations- und Installationshandbuch

Die *{baseUrl}* wird mit ihrem Host und ohne geschweifte Klammern ersetzt.

Die folgende Anleitung erklärt wie man WDB installiert.

**Vorbereitung**:

Auf dem Gerät muss folgendes installiert und wenn möglich, gestartet sein.

Java jdk 1.8.0\_131

Mysql-server

Einen Client

Wdb.jar

Das Betriebssystem ist für die Applikation irrelevant.

**Datenbank**

Mithilfe des wdbinit.sql script kann die Datenbank auf dem Server eingerichtet werden.

Wenn man die wdbinit.sql Datei ausführt, wird die Datenbank erstellt.

**Applikation**

Die Applikation kann an einem beliebigen Ort abgespeichert werden. Der Zugriff erfolgt immer über diese Adresse: *{baseUrl}.*

## Betriebshandbuch

Applikation starten: java –jar wdb.jar

Applikation neustarten: Zuerst den Prozess killen und danach mit Java –jar wdb.jar die Applikation starten.

Logs: In den Ereignisanzeigen des jeweiligen Betriebssystems nachschauen. Die Applikation legt kein eigene Log-files an.

Backups: Die Applikation bietet von sich aus keine Backup Funktion an. Jedoch kann man mit mysqldump ein Backup der Datenbank erstellen.

# Testspezifikation

## Unit-Test

Wir haben uns entschieden, aufgrund der Zeit keine Unit-Tests zu schreiben. Fehler im Code werden durch gelegentliches exploratives Testen während der Entwicklung gefunden. Zudem darf der Entwickler den Code nur auf Git pushen, wenn er auch korrekt läuft.

## Systemtest

### Testfälle

Die *{baseUrl}* wird mit ihrem Host und ohne geschweifte Klammern ersetzt.

#### Benutzer registrieren

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Ein neuer Benutzer registriert sich | |
| **Abgedeckte Anwendungsfälle** | Neues Benutzerkonto erstellen | |
| **Ausgangssituation** | Es besteht noch kein solcher Benutzer. | |
| **Vorbereitungsschritte** | 1. Die API ist auf dem Pfad {baseUrl} erreichbar 2. Postman auf dem Testclient gestartet | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Mit Postman {baseUrl}/user aufrufen mit den folgenden Werten im Requestbody:   {„username“:“Testuser“, „password“:“gibbiX12345“} | | HTTP Response: Code 200 |

#### Benutzer anmelden

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Ein neuer Benutzer meldet sich beim System an. | |
| **Abgedeckte Anwendungsfälle** | Anmelden | |
| **Ausgangssituation** | * 1. Es besteht der User mit dem man sich anmelden will Benutzer. | |
| **Vorbereitungsschritte** | 1. Die API ist auf dem Pfad {baseUrl} erreichbar 2. Postman auf dem Testclient gestartet | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Mit Postman {baseUrl}/login aufrufen mit den folgenden Werten im Requestbody: {„username“:“Testuser“, „password“:“gibbiX12345“} | | HTTP Response: Code 200 |

#### Page erstellen

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Ein neuer Benutzer legt eine neue Page an | |
| **Abgedeckte Anwendungsfälle** | Page erstellen | |
| **Ausgangssituation** | Man ist mit einem Benutzer bei der API eingeloggt | |
| **Vorbereitungsschritte** | 1. Die API ist auf dem Pfad {baseUrl} erreichbar 2. Postman auf dem Testclient gestartet | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Mit Postman {baseUrl}/page aufrufen mit den folgenden Werten im Requestbody:   {„title“:“page title“,  „content“:“hier drin steht der Eintrag der Page“,  authorId: 1,  tagIds: 1} | | HTTP Response: Code 200 |

#### Tags hinzufügen

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Ein Benutzer legt einen Tag an | |
| **Abgedeckte Anwendungsfälle** | Tags hinzufügen | |
| **Ausgangssituation** | Der Tag besteht noch nicht | |
| **Vorbereitungsschritte** | 1. Die API ist auf dem Pfad {baseUrl} erreichbar 2. Postman auf dem Testclient gestartet | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Mit Postman {baseUrl}/tag aufrufen mit den folgenden Werten im Requestbody:   {„Label“} | | HTTP Response: Code 200 |

#### Tags bearbeiten

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Ein Benutzer bearbeitet einen Tag | |
| **Abgedeckte Anwendungsfälle** | Tags bearbeiten | |
| **Ausgangssituation** | Der Tag zum Bearbeiten besteht bereits | |
| **Vorbereitungsschritte** | 1. Die API ist auf dem Pfad {baseUrl} erreichbar 2. Postman auf dem Testclient gestartet | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Mit Postman {baseUrl}/tag/{id} aufrufen mit den folgenden Werten im Requestbody:   {„Label“} | | HTTP Response: Code 200 |

#### Page bearbeiten

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Ein Benutzer bearbeitet eine Page | |
| **Abgedeckte Anwendungsfälle** | Page bearbeiten | |
| **Ausgangssituation** | Die Page zum Bearbeiten besteht bereits | |
| **Vorbereitungsschritte** | 1. Die API ist auf dem Pfad {baseUrl} erreichbar 2. Postman auf dem Testclient gestartet | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Mit Postman {baseUrl}/page/{id} aufrufen mit den folgenden Werten im Requestbody:   {„title“:“changed title“,  „content“:“content anpassen“} | | HTTP Response: Code 200 |

#### Page suchen

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Ein Benutzer sucht nach einer oder mehreren bestimmten Pages. | |
| **Abgedeckte Anwendungsfälle** | Page suchen | |
| **Ausgangssituation** | Applikation läuft | |
| **Vorbereitungsschritte** | 1. Die API ist auf dem Pfad {baseUrl} erreichbar 2. Postman auf dem Testclient gestartet | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Die suche wird im UI definiert und durch verschiedene API-Aufrufe (Je nach Kriterium) durchgeführt. Darum ist diese Funktion nicht automatisiert testbar. | | Kein Resultat erwartet |

#### Page löschen

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Der Benutzer löscht eine Page | |
| **Abgedeckte Anwendungsfälle** | Page löschen | |
| **Ausgangssituation** | Der Benutzer hat bereits eine Page | |
| **Vorbereitungsschritte** | 1. Die API ist auf dem Pfad {baseUrl} erreichbar 2. Postman auf dem Testclient gestartet | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Mit Postman {baseUrl}/page/id?=1 aufrufen | | HTTP Response: Code 200 |

#### Page exportieren

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Der Benutzer exportiert eine Page | |
| **Abgedeckte Anwendungsfälle** | Page exportieren | |
| **Ausgangssituation** | Der Benutzer hat bereits eine Page | |
| **Vorbereitungsschritte** | 1. Die API ist auf dem Pfad {baseUrl} erreichbar 2. Postman auf dem Testclient gestartet | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Mit Postman {baseUrl}/page/id?=1 aufrufen | | HTTP Response: Code 200 |

#### Passwort ändern

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Der Benutzer ändert das Passwort | |
| **Abgedeckte Anwendungsfälle** | Passwort ändern | |
| **Ausgangssituation** | Der Benutzer besitzt ein Account | |
| **Vorbereitungsschritte** | 1. Die API ist auf dem Pfad {baseUrl} erreichbar 2. Postman auf dem Testclient gestartet | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Mit Postman {baseUrl}/user/id aufrufen mit den folgenden Werten im Requestbody:   {„oldPassword“:„gibbiX12345“,  „newPassword“:„admin“} | | HTTP Response: Code 200 |

#### Benutzer deaktivieren

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Der Benutzer deaktiviert seinen Account | |
| **Abgedeckte Anwendungsfälle** | Benutzer deaktivieren (löschen) | |
| **Ausgangssituation** | Der Benutzer besitzt ein Account | |
| **Vorbereitungsschritte** | 1. Die API ist auf dem Pfad {baseUrl} erreichbar 2. Postman auf dem Testclient gestartet | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Mit Postman {baseUrl}/user aufrufen mit den folgenden Werten im Requestbody:   {„password“:„admin“} | | HTTP Response: Code 200 |

#### Benutzer reaktivieren

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Der Benutzer reaktiviert seinen Account | |
| **Abgedeckte Anwendungsfälle** | Benutzer reaktivieren | |
| **Ausgangssituation** | Der Benutzer hat sein Account deaktiviert (gelöscht) | |
| **Vorbereitungsschritte** | 1. Die API ist auf dem Pfad {baseUrl} erreichbar 2. Postman auf dem Testclient gestartet | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Mit Postman {baseUrl}/login aufrufen mit den folgenden Werten im Requestbody:   {„username“:„Testuser“,  „password“:„admin“} | | HTTP Response: Code 200 |

# Testprotokoll

## Systemtest 1

Getestete Version: 1.0

Tester: Miro Albrecht

Datum, Zeit: 12.12.2017 21:00 Uhr

### Testfall 1 „Neues Benutzerkonto erstellen “

Test: **ERFÜLLT**

Bemerkung: -

### Testfall 2 „Anmelden“

Test: **ERFÜLLT**

Bemerkung: -

### Testfall 3 „Page erstellen“

Test: **ERFÜLLT**

Bemerkung: -

### Testfall 4 „Tags hinzufügen“

Test: **ERFÜLLT**

Bemerkung: -

### Testfall 5 „Tags bearbeiten“

Test: **ERFÜLLT**

Bemerkung: -

### Testfall 6 „Page bearbeiten“

Test: **ERFÜLLT**

Bemerkung: -

### Testfall 7 „Page suchen“

Test: **ERFÜLLT**

Bemerkung: -

### Testfall 8 „Page löschen“

Test: **ERFÜLLT**

Bemerkung: -

### Testfall 9 „Page exportieren“

Test: **ERFÜLLT**

Bemerkung: -

### Testfall 10 „Passwort ändern“

Test: **ERFÜLLT**

Bemerkung: -

### Testfall 11 „Benutzer deaktivieren (löschen)“

Test: **ERFÜLLT**

Bemerkung: -

### Testfall 12 „Benutzer reaktivieren“

Test: **ERFÜLLT**

Bemerkung: -

# Projektplanung

Siehe Projektplan, Version 3.0 vom 12.12.2017

# Anhang A: Quellcode

## API

Controller

package api;  
  
import org.springframework.stereotype.Component;  
  
import javax.persistence.EntityManager;  
import javax.persistence.PersistenceContext;  
  
  
@Component  
public class Controller {  
   
 @PersistenceContext  
 protected EntityManager em;  
  
 public Controller(EntityManager em) {  
 this.em = em;  
 }  
}

Page Controller

package api;  
  
import entity.data.DataPage;  
import entity.data.DataTag;  
import entity.domain.Attachment;  
import entity.domain.Page;  
import entity.domain.Tag;  
import entity.domain.User;  
import org.springframework.transaction.annotation.Transactional;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import javax.persistence.EntityManager;  
import javax.persistence.Query;  
import javax.servlet.http.HttpSession;  
import java.util.ArrayList;  
import java.util.List;  
  
import static org.springframework.web.bind.annotation.RequestMethod.\*;  
  
  
@RestController  
@RequestMapping("/page")  
public class PageController extends Controller  
{  
   
 public PageController (EntityManager em)  
 {  
 super(em);  
 }  
   
 @RequestMapping(method = GET)  
 public DataPage[] pages ()  
 {  
 Query query = em.createNamedQuery("Page.findAll", Page.class);  
   
 if (query.getMaxResults() == 0) return new DataPage[0];  
   
 List<Page> tags = query.getResultList();  
 DataPage[] dataPages = new DataPage[tags.size()];  
 for (int i = 0; i < tags.size(); i++)  
 {  
 dataPages[i] = dataFromDomain(tags.get(i));  
 }  
 return dataPages;  
 }  
   
 public static DataPage dataFromDomain (Page page)  
 {  
 List<DataTag> tags = new ArrayList<>();  
 page.getTags().forEach(tag -> tags.add(TagController.dataFromDomain(tag)));  
   
   
 return new DataPage(page.getTitle(), page.getContent(), tags.toArray(new DataTag[0]), page  
 .getAuthor().getId());  
 }  
   
 @RequestMapping(method = GET, value = "/{pageId}")  
 public DataPage getPage (@PathVariable long pageId)  
 {  
 return dataFromDomain(em.createNamedQuery("Page.findPageById", Page.class)  
 .setParameter(1, pageId)  
 .getSingleResult());  
 }  
   
 @RequestMapping(method = POST)  
 @Transactional  
 public void postPage (@RequestBody NewPage newPage, HttpSession session) throws Exception  
 {  
 User user = ((User) session.getAttribute("User"));  
   
 if (user == null)  
 {  
 throw new Exception("User not logged in");  
 }  
   
 List<Tag> tags = new ArrayList<>();  
   
 for (long id : newPage.tagIds)  
 {  
 tags.add(em.find(Tag.class, id));  
 }  
   
 Page page = new Page(newPage.title, newPage.content, tags, user);  
 em.persist(page);  
 }  
   
 @RequestMapping(method = POST, value = "/{pageId}/tag")  
 @Transactional  
 public void addTagToPage (@PathVariable long pageId, @RequestBody long tagId) throws Exception  
 {  
 Page page = em.find(Page.class, pageId);  
 Tag tag = em.find(Tag.class, tagId);  
   
 if (page != null && tag != null)  
 {  
 page.getTags().add(tag);  
 em.merge(page);  
 }  
 else  
 {  
 throw new Exception("Cannot add tag to page. Please check the ids");  
 }  
 }  
   
   
 @RequestMapping(method = PUT, value = "/{pageId}")  
 @Transactional  
 public void putPage (  
 @PathVariable long pageId, @RequestBody ChangePage newPage, HttpSession session) throws Exception  
 {  
 User user = ((User) session.getAttribute("User"));  
   
 if (user == null)  
 {  
 throw new Exception("User not logged in");  
 }  
   
 Page page = em.find(Page.class, pageId);  
   
 if (page == null)  
 {  
 throw new Exception("Page nicht gefunden");  
 }  
   
 if (newPage.getTitle() != null) page.setTitle(newPage.getTitle());  
   
 if (newPage.getContent() != null) page.setContent(newPage.getContent());  
   
 em.merge(page);  
 }  
   
 @RequestMapping(method = DELETE, value = "/{pageId}")  
 @Transactional  
 public void deletePage (@PathVariable long pageId, HttpSession session) throws Exception  
 {  
 User user = ((User) session.getAttribute("User"));  
   
 if (user == null)  
 {  
 throw new Exception("User not logged in");  
 }  
   
 Page page = em.find(Page.class, pageId);  
   
 if (page == null)  
 {  
 throw new Exception("Page nicht gefunden");  
 }  
   
 em.remove(page);  
 }  
   
 @RequestMapping(method = DELETE, value = "/{pageId}/tag")  
 @Transactional  
 public void removeTagFromPage (  
 @PathVariable long pageId, @RequestBody long tagId, HttpSession session) throws Exception  
 {  
 User user = ((User) session.getAttribute("User"));  
   
 if (user == null)  
 {  
 throw new Exception("User not logged in");  
 }  
   
 Page page = em.find(Page.class, pageId);  
 Tag tag = em.find(Tag.class, tagId);  
   
 if (page == null || tag == null)  
 {  
 throw new Exception("Page oder Tag nicht gefunden");  
 }  
   
 page.getTags().remove(tag);  
   
 em.merge(page);  
 }  
   
 @RequestMapping(method = DELETE, value = "/{pageId}/attachment")  
 String removeAttFromPage (@PathVariable String pageId, @RequestBody long attachmentId)  
 {  
 return String.format("Remove attachment to page with id %s, %d", pageId, attachmentId);  
 }  
   
 @RequestMapping(method = POST, value = "/{pageId}/attachment")  
 String addAttToPage (@PathVariable String pageId, @RequestBody Attachment attachment)  
 {  
 return String.format("Add attachment to page with id %s, %s", pageId, attachment.toString());  
 }  
}  
  
class NewPage extends ChangePage  
{  
 long authorId;  
 long[] tagIds;  
   
 public long getAuthorId ()  
 {  
 return authorId;  
 }  
   
 public void setAuthorId (long authorId)  
 {  
 this.authorId = authorId;  
 }  
   
 public long[] getTagIds ()  
 {  
 return tagIds;  
 }  
   
 public void setTagIds (long[] tagIds)  
 {  
 this.tagIds = tagIds;  
 }  
}  
  
class ChangePage  
{  
 String title;  
 String content;  
   
 public String getTitle ()  
 {  
 return title;  
 }  
   
 public void setTitle (String title)  
 {  
 this.title = title;  
 }  
   
 public String getContent ()  
 {  
 return content;  
 }  
   
 public void setContent (String content)  
 {  
 this.content = content;  
 }  
   
}

Session Controller

package api;  
  
  
import entity.domain.User;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.crypto.password.StandardPasswordEncoder;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
import sun.rmi.runtime.Log;  
  
import javax.persistence.EntityManager;  
import javax.servlet.http.HttpSession;  
  
import static org.springframework.web.bind.annotation.RequestMethod.POST;  
  
@RestController  
public class SessionController extends Controller  
{  
 @Autowired  
 StandardPasswordEncoder passwordEncoder;  
   
 public SessionController (EntityManager em)  
 {  
 super(em);  
 }  
   
 @RequestMapping(method = POST, value = "/login")  
 public void login(@RequestBody Login login, HttpSession session) throws Exception  
 {  
 if (login == null)  
 {  
 return;  
 }  
   
 if (session.getAttribute("User") != null)  
 {  
 throw new Exception("Already logged in");  
 }  
   
 User user = em.createNamedQuery("User.findUserByUsername", User.class).setParameter(1, login.getUsername()).getSingleResult();  
   
 if (user == null)  
 {  
 throw new Exception("Invalid credentials");  
 }  
   
 if (passwordEncoder.matches(login.getPassword(), user.getPassword()))  
 {  
 session.setAttribute("User", user);  
 }  
 else  
 {  
 throw new Exception("Invalid credentials");  
 }  
 }  
   
 @RequestMapping(method = POST, value = "/logout")  
 public void logout(HttpSession session)  
 {  
 session.removeAttribute("User");  
 }  
}  
  
class Login  
{  
 private String username;  
 private String password;  
   
 public String getUsername ()  
 {  
 return username;  
 }  
   
 public void setUsername (String username)  
 {  
 this.username = username;  
 }  
   
 public String getPassword ()  
 {  
 return password;  
 }  
   
 public void setPassword (String password)  
 {  
 this.password = password;  
 }

}

Tag Controller

package api;  
  
import entity.data.DataPage;  
import entity.data.DataTag;  
import entity.domain.Page;  
import entity.domain.Tag;  
import org.hibernate.type.ListType;  
import org.springframework.transaction.annotation.Transactional;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import javax.persistence.EntityManager;  
import javax.persistence.Query;  
import java.util.ArrayList;  
import java.util.List;  
  
import static org.springframework.web.bind.annotation.RequestMethod.\*;  
  
  
@RestController  
@RequestMapping("/tag")  
public class TagController extends Controller  
{  
 public TagController (EntityManager em)  
 {  
 super(em);  
 }  
   
 @RequestMapping(method = GET)  
 DataTag[] getTags ()  
 {  
 Query query = em.createNamedQuery("Tag.findAll", Tag.class);  
   
 if (query.getMaxResults() == 0) return new DataTag[0];  
   
 List<Tag> tags = query.getResultList();  
 DataTag[] dataDataTags = new DataTag[tags.size()];  
 for (int i = 0; i < tags.size(); i++)  
 {  
 dataDataTags[i] = dataFromDomain(tags.get(i));  
 }  
 return dataDataTags;  
 }  
   
 public static DataTag dataFromDomain (Tag tag)  
 {  
 DataTag dataTag = new DataTag(tag.getName());  
 dataTag.setId(tag.getId());  
 return dataTag;  
 }  
   
 @RequestMapping(method = GET, value = "/{tagId}")  
 DataTag getTag (@PathVariable long tagId)  
 {  
 Query query = em.createNamedQuery("Tag.findTagById", Tag.class);  
 query.setParameter(1, tagId);  
 Tag tag = (Tag) query.getSingleResult();  
 return dataFromDomain(tag);  
 }  
   
 @RequestMapping(method = GET, value = "/{tagId}/page")  
 public DataPage[] getPagesFromTag (@PathVariable long tagId) throws Exception  
 {  
 Tag tag = em.find(Tag.class, tagId);  
   
 if (tag == null)  
 {  
 throw new Exception("Tag nicht gefunden");  
 }  
   
 List<Page> pagesWithTag = new ArrayList<>();  
   
 List<Page> pages = em.createNamedQuery("Page.findAll", Page.class).getResultList();  
 pages.forEach(page -> {  
 if (page.getTags().contains(tag))  
 {  
 pagesWithTag.add(page);  
 }  
 });  
   
 DataPage[] dataPages = new DataPage[pagesWithTag.size()];  
   
 for (int i = 0; i < pagesWithTag.size(); i++)  
 {  
 dataPages[i] = PageController.dataFromDomain(pagesWithTag.get(i));  
 }  
   
 return dataPages;  
 }  
   
 @RequestMapping(method = POST)  
 @Transactional  
 public void createTag (@RequestBody String label)  
 {  
 Tag tag = new Tag(label);  
 em.persist(tag);  
 }  
   
 @RequestMapping(method = PUT, value = "/{tagId}")  
 @Transactional  
 public void updateTag (@PathVariable long tagId, @RequestBody String label)  
 {  
 Tag tag = em.find(Tag.class, tagId);  
 tag.setName(label);  
 em.merge(tag);  
 }  
   
 @RequestMapping(method = DELETE, value = "/{tagId}")  
 @Transactional  
 public void deleteTag (@PathVariable long tagId)  
 {  
 Tag tag = em.find(Tag.class, tagId);  
 em.remove(tag);  
 }  
}

User Controller

package api;  
  
import entity.data.DataPage;  
import entity.data.DataUser;  
import entity.domain.Page;  
import entity.domain.User;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.security.crypto.password.StandardPasswordEncoder;  
import org.springframework.transaction.annotation.Transactional;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import javax.persistence.EntityManager;  
import javax.persistence.Query;  
import javax.servlet.http.HttpSession;  
import java.util.List;  
  
import static org.springframework.web.bind.annotation.RequestMethod.\*;  
  
  
@RestController  
@RequestMapping("/user")  
public class UserController extends Controller  
{  
 @Autowired  
 private StandardPasswordEncoder passwordEncoder;  
   
 @Bean  
 public static StandardPasswordEncoder standardPasswordEncoder() {  
 return new StandardPasswordEncoder();  
 }  
   
 public UserController (EntityManager em)  
 {  
 super(em);  
 }  
   
 @RequestMapping(method = GET)  
 DataUser[] getUsers ()  
 {  
 Query query = em.createNamedQuery("User.findAllUserByActive", User.class);  
 query.setParameter(1, true);  
   
 if (query.getMaxResults() == 0) return new DataUser[0];  
   
 List<User> userList = query.getResultList();  
 DataUser[] dataUsers = new DataUser[userList.size()];  
   
 for (int i = 0; i < userList.size(); i++)  
 {  
 dataUsers[i] = dataFromDomain(userList.get(i));  
 }  
   
 return dataUsers;  
 }  
   
 public static DataUser dataFromDomain (User user)  
 {  
 return new DataUser(user.getId(), user.getUsername(), user.isActive());  
 }  
   
 @RequestMapping(method = GET, value = "/{userId}/page")  
 public DataPage[] getPages(@PathVariable long userId)  
 {  
 User user = em.find(User.class, userId);  
   
 Page[] pages = em.createNamedQuery("Page.findAllPageByAuthorId", Page.class).setParameter(1, user).getResultList().toArray(new Page[0]);  
   
 DataPage[] dataPages = new DataPage[pages.length];  
   
 for (int i = 0; i < pages.length; i++)  
 {  
 dataPages[i] = PageController.dataFromDomain(pages[i]);  
 }  
   
 return dataPages;  
 }  
   
 @RequestMapping(method = GET, value = "/{userId}")  
 DataUser getUserById (@PathVariable long userId)  
 {  
 Query query = em.createNamedQuery("User.findUserById");  
 query.setParameter(1, userId);  
   
 User user = ((User) query.getSingleResult());  
   
 return dataFromDomain(user);  
 }  
   
 @RequestMapping(method = POST)  
 @Transactional  
 public void createUser (@RequestBody NewUser newUser)  
 {  
 if (newUser == null)  
 {  
 return;  
 }  
   
 User user = new User(newUser.getUsername(), passwordEncoder.encode(newUser.getPassword()));  
 em.persist(user);  
 }  
   
 @RequestMapping(method = PUT, value = "/{userId}")  
 @Transactional  
 public void updatePassword (  
 @PathVariable long userId,  
 @RequestBody PasswordChange pw, HttpSession session) throws Exception  
 {  
 User user = ((User) session.getAttribute("User"));  
   
 if (user == null)  
 {  
 throw new Exception("User not logged in");  
 }  
   
 Query query = em.createNamedQuery("User.findUserById");  
 query.setParameter(1, userId);  
   
 User user1 = ((User) query.getSingleResult());  
   
 if (passwordEncoder.matches(pw.getOldPassword(), user.getPassword()) && user.getId() == user1.getId())  
 {  
 user.setPassword(passwordEncoder.encode(pw.getNewPassword()));  
 em.merge(user);  
   
 session.setAttribute("User", user);  
 }  
 else  
 {  
 throw new Exception("Wrong password");  
 }  
 }  
   
 @RequestMapping(method = DELETE, value = "/")  
 @Transactional  
 public void deleteUser(@RequestBody String password, HttpSession session) throws Exception  
 {  
 User user = ((User) session.getAttribute("User"));  
   
 if (user == null)  
 {  
 throw new Exception("User not logged in");  
 }  
   
 if (passwordEncoder.matches(password, user.getPassword()))  
 {  
 Query query = em.createNamedQuery("User.findUserById");  
 query.setParameter(1, user.getId());  
   
 User user1 = ((User) query.getSingleResult());  
 em.remove(user1);  
 session.removeAttribute("User");  
 }  
 else  
 {  
 throw new Exception("Wrong password");  
 }  
 }  
}  
  
class NewUser  
{  
 private String username;  
 private String password;  
   
 public String getUsername ()  
 {  
 return username;  
 }  
   
 public void setUsername (String username)  
 {  
 this.username = username;  
 }  
   
 public String getPassword ()  
 {  
 return password;  
 }  
   
 public void setPassword (String password)  
 {  
 this.password = password;  
 }  
}  
  
class PasswordChange  
{  
 private String oldPassword;  
 private String newPassword;  
   
 public String getOldPassword ()  
 {  
 return oldPassword;  
 }  
   
 public void setOldPassword (String oldPassword)  
 {  
 this.oldPassword = oldPassword;  
 }  
   
 public String getNewPassword ()  
 {  
 return newPassword;  
 }  
   
 public void setNewPassword (String newPassword)  
 {  
 this.newPassword = newPassword;  
 }  
}

## Connection

Connector

package connection;  
  
import javax.persistence.EntityManager;  
import javax.persistence.EntityManagerFactory;  
import javax.persistence.Persistence;  
  
public class Connector {  
  
 EntityManagerFactory entityManagerFactory = null;  
  
 public Connector(String persistenceUnit) {  
 entityManagerFactory = Persistence.createEntityManagerFactory(persistenceUnit);  
 }  
  
 public EntityManager getEntityManagerInstance() {  
 return entityManagerFactory.createEntityManager();  
 }  
}

## Entity

### Data

Attachement

package entity.data;  
  
import java.util.Date;  
  
  
public class DataAttachment  
{  
 private long id;  
 private String fileName;  
 private long size;  
 private Date date;  
   
 public DataAttachment (String fileName)  
 {  
 this.fileName = fileName;  
 }  
   
 public DataAttachment (String fileName, long size, Date date)  
 {  
 this.fileName = fileName;  
 this.size = size;  
 this.date = date;  
 }  
   
 public long getId ()  
 {  
 return id;  
 }  
   
 public void setId (long id)  
 {  
 this.id = id;  
 }  
   
 public String getFileName ()  
 {  
 return fileName;  
 }  
   
 public void setFileName (String fileName)  
 {  
 this.fileName = fileName;  
 }  
   
 public long getSize ()  
 {  
 return size;  
 }  
   
 public void setSize (long size)  
 {  
 this.size = size;  
 }  
   
 public Date getDate ()  
 {  
 return date;  
 }  
   
 public void setDate (Date date)  
 {  
 this.date = date;  
 }  
}

Data Page

package entity.data;  
  
public class DataPage  
{  
 private long id;  
 private String title;  
 private String content;  
 private DataTag[] dataTags;  
 private DataAttachment[] dataAttachments;  
 private long author;  
   
 public DataPage () {}  
   
 public DataPage (String title, String content, DataTag[] dataTags, long author)  
 {  
 this.title = title;  
 this.content = content;  
 this.dataTags = dataTags;  
 this.author = author;  
 }  
   
 public long getId ()  
 {  
 return id;  
 }  
   
 public void setId (long id)  
 {  
 this.id = id;  
 }  
   
 public String getTitle ()  
 {  
 return title;  
 }  
   
 public void setTitle (String title)  
 {  
 this.title = title;  
 }  
   
 public String getContent ()  
 {  
 return content;  
 }  
   
 public void setContent (String content)  
 {  
 this.content = content;  
 }  
   
 public DataTag[] getDataTags ()  
 {  
 return dataTags;  
 }  
   
 public void setDataTags (DataTag[] dataTags)  
 {  
 this.dataTags = dataTags;  
 }  
   
 public DataAttachment[] getDataAttachments ()  
 {  
 return dataAttachments;  
 }  
   
 public void setDataAttachments (DataAttachment[] dataAttachments)  
 {  
 this.dataAttachments = dataAttachments;  
 }  
   
 public long getAuthor ()  
 {  
 return author;  
 }  
   
 public void setAuthor (long author)  
 {  
 this.author = author;  
 }  
}

Data Tag

package entity.data;  
  
public class DataTag  
{  
 private long id;  
 private String label;  
   
 public DataTag (String label)  
 {  
 this.label = label;  
 }  
   
 public long getId ()  
 {  
 return id;  
 }  
   
 public void setId (long id)  
 {  
 this.id = id;  
 }  
   
 public String getLabel ()  
 {  
 return label;  
 }  
   
 public void setLabel (String label)  
 {  
 this.label = label;  
 }  
}

Data User

package entity.data;  
  
public class DataUser  
{  
 private long id;  
 private String username;  
 private boolean active;  
 private long[] pageIds;  
   
 public DataUser (String username)  
 {  
 this.username = username;  
 }  
   
 public DataUser (long id, String username, boolean active)  
 {  
 this.id = id;  
 this.username = username;  
 this.active = active;  
 }  
   
 public DataUser (String username, boolean active)  
 {  
 this.username = username;  
 this.active = active;  
 }  
   
 private long getId ()  
 {  
 return id;  
 }  
   
 private void setId (long id)  
 {  
 this.id = id;  
 }  
   
 private long[] getPageIds ()  
 {  
 return pageIds;  
 }  
   
 private void setPageIds (long[] pageIds)  
 {  
 this.pageIds = pageIds;  
 }  
   
 public String getUsername ()  
 {  
 return username;  
 }  
   
 public void setUsername (String username)  
 {  
 this.username = username;  
 }  
   
 public boolean isActive ()  
 {  
 return active;  
 }  
   
 public void setActive (boolean active)  
 {  
 this.active = active;  
 }  
}

### Domain

Attachement

package entity.domain;  
  
import javax.persistence.\*;  
import java.util.Date;  
  
@Entity  
@NamedQueries({  
 @NamedQuery(name = "Attachment.findAttachmentById", query = "SELECT a FROM Attachment AS a WHERE a.id = ?1"),  
 @NamedQuery(name= "Attachment.findAttachmentByDate", query = "SELECT a FROM Attachment AS a WHERE a.date = ?1")  
})  
public class Attachment  
{  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 private long id;  
 private String fileName;  
 private long size;  
 private Date date;  
  
 public Attachment() {  
 }  
  
 public long getId() {  
 return id;  
 }  
 public void setId(long id) {  
 this.id = id;  
 }  
 public String getFileName() {  
 return fileName;  
 }  
 public void setFileName(String fileName) {  
 this.fileName = fileName;  
 }  
 public long getSize() {  
 return size;  
 }  
 public void setSize(long size) {  
 this.size = size;  
 }  
 public Date getDate() {  
 return date;  
 }  
 public void setDate(Date date) {  
 this.date = date;  
 }  
}

Page

package entity.domain;  
  
import javax.persistence.\*;  
import java.util.List;  
  
  
@Entity  
@NamedQueries({  
 @NamedQuery(name = "Page.findPageById", query = "SELECT p FROM Page AS p WHERE p.id = ?1"),  
 @NamedQuery(name = "Page.findPageByTitle", query = "SELECT p FROM Page AS p WHERE p.title LIKE '?1%'"),  
 @NamedQuery(name = "Page.findAllPageByAuthorId", query = "SELECT p FROM Page AS p WHERE p.author = ?1"),  
 @NamedQuery(name = "Page.findAll", query = "SELECT p FROM Page AS p")  
})  
public class Page  
{  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 private long id;  
 private String title;  
 private String content;  
   
 @ManyToMany  
 private List<Tag> tags;  
   
 @ManyToOne(targetEntity = User.class)  
 @JoinColumn(name="author\_id")  
 private User author;  
   
 public Page ()  
 {  
 }  
   
 public Page (String title, String content, List<Tag> tags, User author)  
 {  
 this.id = id;  
 this.title = title;  
 this.content = content;  
 this.tags = tags;  
 this.author = author;  
 }  
   
 public long getId ()  
 {  
 return id;  
 }  
   
 public void setId (long id)  
 {  
 this.id = id;  
 }  
   
 public String getTitle ()  
 {  
 return title;  
 }  
   
 public void setTitle (String title)  
 {  
 this.title = title;  
 }  
   
 public String getContent ()  
 {  
 return content;  
 }  
   
 public void setContent (String content)  
 {  
 this.content = content;  
 }  
   
 @JoinTable(name = "tag\_page", joinColumns =  
 @JoinColumn(name = "pageId", referencedColumnName="id"),  
 inverseJoinColumns = @JoinColumn(name = "tagId", referencedColumnName="id"))  
 public List<Tag> getTags ()  
 {  
 return tags;  
 }  
   
 public void setTags (List<Tag> tags)  
 {  
 this.tags = tags;  
 }  
   
 public User getAuthor ()  
 {  
 return author;  
 }  
   
 public void setAuthor (User author)  
 {  
 this.author = author;  
 }  
}

Tag

package entity.domain;  
  
  
import javax.persistence.\*;  
import java.util.List;  
import java.util.Objects;  
  
  
@Entity  
@NamedQueries({@NamedQuery(name = "Tag.findTagById",  
 query = "SELECT t FROM Tag AS t WHERE t.id = ?1"), @NamedQuery(name = "Tag.findTagByName",  
 query = "SELECT t FROM Tag AS t WHERE t.name = ?1"), @NamedQuery(name = "Tag.findAll",  
 query = "SELECT t FROM Tag AS t")})  
public class Tag  
{  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 private long id;  
 private String name;  
   
 public List<Page> getPages ()  
 {  
 return pages;  
 }  
   
 public void setPages (List<Page> pages)  
 {  
 this.pages = pages;  
 }  
   
 @ManyToMany(fetch = FetchType.LAZY, mappedBy = "tags")  
 private List<Page> pages;  
   
 public Tag () {}  
   
 public Tag (String name)  
 {  
 this.name = name;  
 }  
   
 public long getId ()  
 {  
 return id;  
 }  
   
 public void setId (long id)  
 {  
 this.id = id;  
 }  
   
 public String getName ()  
 {  
 return name;  
 }  
   
 public void setName (String name)  
 {  
 this.name = name;  
 }  
   
 @Override  
 public boolean equals (Object o)  
 {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Tag tag = (Tag) o;  
 return id == tag.id;  
 }  
   
 @Override  
 public int hashCode ()  
 {  
   
 return Objects.hash(id);  
 }  
}

User

package entity.domain;  
  
import javax.persistence.\*;  
import java.util.List;  
  
  
@Entity  
@NamedQueries({@NamedQuery(name = "User.findUserById", query = "SELECT u FROM User AS u WHERE u.id = ?1"), @NamedQuery(  
 name = "User.findUserByUsername",  
 query = "SELECT u FROM User AS u WHERE u.username = ?1"), @NamedQuery(name = "User.findAllUserByActive",  
 query = "SELECT u FROM User AS u WHERE u.active = ?1")})  
public class User  
{  
 @Id  
 @GeneratedValue(strategy = GenerationType.AUTO)  
 private long id;  
 @Column(unique = true)  
 private String username;  
 private String password;  
 private boolean active;  
   
 @OneToMany(mappedBy = "author")  
 private List<Page> pages;  
 public User(){}  
   
 public User (String username, String password)  
 {  
 this.username = username;  
 this.password = password;  
 this.active = true;  
 }  
   
 public long getId ()  
 {  
 return id;  
 }  
   
 public void setId (long id)  
 {  
 this.id = id;  
 }  
   
 public String getUsername ()  
 {  
 return username;  
 }  
   
 public void setUsername (String username)  
 {  
 this.username = username;  
 }  
   
 public String getPassword ()  
 {  
 return password;  
 }  
   
 public void setPassword (String password)  
 {  
 this.password = password;  
 }  
   
 public boolean isActive ()  
 {  
 return active;  
 }  
   
 public void setActive (boolean active)  
 {  
 this.active = active;  
 }  
}

## WDB

WdbApplication

package wdb.wdb;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.boot.autoconfigure.domain.EntityScan;  
import org.springframework.context.annotation.ComponentScan;  
  
  
@SpringBootApplication  
@ComponentScan(basePackages = {"api", "connection"})  
@EntityScan(basePackages = {"entity"})  
public class WdbApplication  
{  
 public static void main (String[] args)  
 {  
 SpringApplication.run(WdbApplication.class, args);  
 }  
}

## Resources

Application Properties

#https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-sql.html  
spring.datasource.url=jdbc:mysql://localhost/test  
spring.datasource.username=root  
spring.datasource.password=  
spring.datasource.driver-class-label=com.mysql.jdbc.Driver  
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect  
spring.jpa.show-sql=true

Project Object Model

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>wdb</groupId>  
 <artifactId>wdb</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>wdb</name>  
 <description>Projekt Wissensdatenbank WDB</description>  
  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.5.8.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  
 <java.version>1.8</java.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-configuration-processor</artifactId>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.restdocs</groupId>  
 <artifactId>spring-restdocs-mockmvc</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.security</groupId>  
 <artifactId>spring-security-web</artifactId>  
 <version>5.0.1.BUILD-SNAPSHOT</version>  
 </dependency>  
  
 </dependencies>  
 <repositories>  
 <repository>  
 <id>spring-snapshots</id>  
 <name>Spring Snapshots</name>  
 <url>https://repo.spring.io/libs-snapshot</url>  
 <snapshots>  
 <enabled>true</enabled>  
 </snapshots>  
 </repository>  
 </repositories>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

SQL initialization

**START TRANSACTION**;  
  
**CREATE Database** wdb;  
**use** wdb;  
-- --------------------------------------------------------  
  
--  
-- Tabellenstruktur für Tabelle `DataPage`  
--  
  
**CREATE TABLE** `page` (  
 `id` **int**(11) **NOT NULL PRIMARY KEY AUTO\_INCREMENT**,  
 `title` **varchar**(255),  
 `content` **varchar**(255),  
 `author\_id` **int**(11)  
);  
  
-- --------------------------------------------------------  
  
--  
-- Tabellenstruktur für Tabelle `DataTag`  
--  
  
**CREATE TABLE** `tag` (  
 `id` **int**(11) **NOT NULL PRIMARY KEY AUTO\_INCREMENT**,  
 `name` **varchar**(255)  
);  
  
-- --------------------------------------------------------  
  
--  
-- Tabellenstruktur für Tabelle `User`  
--  
  
**CREATE TABLE** `user` (  
 `id` **int**(11) **NOT NULL PRIMARY KEY AUTO\_INCREMENT**,  
 `username` **varchar**(255) **UNIQUE**,  
 `password` **varchar**(255),  
 `active` **tinyint**(1)  
);  
  
  
  
**CREATE TABLE** `tag\_page` (  
 `tagId` **INT**(11) **NOT NULL**,  
 `pageId` **INT**(11) **NOT NULL**,  
 **PRIMARY KEY** (`tagId`, `pageId`),  
 **CONSTRAINT** `FK\_Tag` **FOREIGN KEY** (`tagId`) **REFERENCES** `tag` (`id`),  
 **CONSTRAINT** `FK\_Page` **FOREIGN KEY** (`pageId`) **REFERENCES** `page` (`id`)  
);  
--  
-- Indizes der exportierten Tabellen  
--  
  
--  
-- Indizes für die Tabelle `DataPage`  
--  
**ALTER TABLE** `page`  
 **ADD KEY** `fk\_user` (`author\_id`);  
  
**ALTER TABLE** `page`  
 **ADD CONSTRAINT** `fk\_user` **FOREIGN KEY** (`author\_id`) **REFERENCES** `user` (`id`);  
**COMMIT**;