Realisierungsbericht   
Wissensdatenbank

| Auftraggeber | Georg Ninck |
| --- | --- |
| Projektleiter | Joel Häberli |
| Autor | Joel Häberli |
| Klassifizierung | Nicht klassifiziert, ~~Intern, Vertraulich, GEHEIM~~ |
| Status | Zur Prüfung |
|  |  |

Änderungsverzeichnis

| Datum | Version | Änderung | Autor |
| --- | --- | --- | --- |
| 21.11.2017 | 0.1 | Draft | David Schor |
|  |  |  |  |
|  |  |  |  |

Inhaltsverzeichnis

1 Zusammenfassung 3

2 Technische Detailspezifikation 3

2.1 Systemdesign 3

2.1.1 Struktur 3

2.1.2 Dynamik 3

2.2 Schnittstellendefinitionen 4

2.3 Datenmodell 4

2.4 Sicherheit (ISDS) 4

3 Systemdokumentation 5

3.1 Anwendungshandbuch 5

3.1.1 Übersicht 5

3.1.2 Funktionen und Detailbeschreibung 5

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

Anhang B: Testcode Unit-Tests 16

# Zusammenfassung

*Wozu dient das Dokument (Zweck) und welche Informationen enthält es? Geben Sie hier eine kurze Zusammenfassung des Inhalts dieses Dokumentes (das Wichtigste pro Kapitel).*

# Technische Detailspezifikation

## Systemdesign

*Im Konzeptbericht haben Sie die Systemarchitektur entworfen. Dort haben Sie beschrieben aus welchen Elementen (Schichten, Pakete, Klassen, Module) Ihr System grob besteht. Während der Realisierung des Systems verfeinern Sie diese Architektur schrittweise. Weitere Elemente kommen hinzu, andere müssen aufgeteilt, oder anders angeordnet werden.*

*Hier geht es nun darum die am Ende der Realisierung gültige Systemarchitektur und den Systemdesign zu beschreiben. Dazu gehören die statische Struktur des Systems und die Dynamik. Der zweite Punkt ist vor allem bei objektorientierter Umsetzung wichtig.*

### Struktur

*Beschreiben Sie ausgehend von der Systemarchitektur den strukturellen Aufbau ihres Systems. Wenn Sie objektorientiert entwickeln, verwenden Sie dazu UML-Klassendiagramme und Paketdiagramme. Andernfalls verwenden Sie Blockdiagramme, welche die einzelnen Module und ihre Funktionen/Prozeduren, sowie die Aufrufbeziehungen zwischen den Modulen zeigen (siehe auch Beispiele im Konzeptbericht).*

***Hinweis****: Es muss nicht die hinterste und letzte Hilfsklasse oder Hilfsfunktion in den Diagrammen erscheinen. Vielmehr sollen diejenigen Elemente gezeigt werden, welche nötig sind, um das System zu verstehen. Sie müssen Ihr System also auf einer sinnvollen Ebene abstrahieren.*

***Hinweis****: Wenn Sie mit einem Framework oder mit externen Libraries arbeiten, so müssen Sie in der Regel auch Elemente der Framework oder der Library, welche unmittelbar mit Ihrem System zusammenhängen, in Ihre Diagramme aufnehmen. Sonst wird man Ihr System kaum verstehen.*

***Wichtig****: Arbeiten Sie mit mehreren Diagrammen. Geben Sie zu jedem Diagramm eine kurze Beschreibung dessen, was Sie mit dem Diagramm darstellen wollen.*

***Wichtig****: Dieser Abschnitt enthält nicht einfach nur Diagramme. Sie müssen je nachdem auch einzelne Elemente im Text näher beschreiben (Aufgaben des Elementes, Designentscheidungen, …). Dies gilt vor allem für Elemente, deren Zweck nicht direkt aus den Diagrammen hervorgeht. Denken Sie daran, dass Sie dem Leser, z.B. dem IPA-Experten oder der Person, welche Ihr System erweitern soll, erklären wollen, wie Ihr System gebaut ist und wie es arbeitet.*

### Dynamik

*Oft reicht eine rein strukturelle Beschreibung nicht aus, um das System zu verstehen.*

*Illustrieren Sie also hier die wichtigsten Abläufe in Ihrem System z.B. mit Hilfe von Sequenzdiagrammen oder allenfalls Struktogrammen.*

*Bei der Auswahl der zu illustrierenden Abläufe können Sie sich folgende Fragen stellen:*

* *Welches ist der komplexeste Anwendungsfall im System? Illustrieren Sie diesen Ablauf mit einem Sequenzdiagramm. Meist wird es sich dabei um den zentralen Anwendungsfall Ihres Systems handeln.*
* *Gibt es in Ihrem System Schnittstellen zu anderen, allenfalls externen Systemen, wo es darauf ankommt eine bestimmte Meldungsreihenfolge einzuhalten? Oder müssen gar Meldungen auf verschiedenen Schnittstellen koordiniert werden? Setzen Sie auch hier ein Sequenzdiagramm zur Illustration ein.*
* *Gibt es in Ihrem System komplizierte Algorithmen? Hier eignet sich allenfalls ein Struktogramm zur Illustration.*

***Wichtig****: Arbeiten Sie mit mehreren Diagrammen. Geben Sie zu jedem Diagramm eine kurze Beschreibung, dessen, was Sie mit dem Diagramm darstellen wollen.*

***Wichtig****: Auch hier gilt, abstrahieren Sie auf der richtigen Ebene. Generierte Sequenzdiagramme, welche die hinterste und letzte Zeile Code illustrieren sind unnütz.*

## 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.

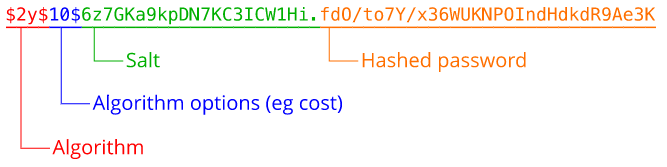
## Datenmodell

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 persönliche 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 keinen 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 die 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 **GM Kevin Colbert**

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:   {„name“:“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:   {„name“:“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“:“{USER\_ID}“, „tagsIds“:“{TAG\_IDS}“} | | 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“:“taglabel“} | | 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“:“taglabelchange“} | | 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. | | HTTP Response: Code 200 |

#### 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 aufrufen mit den folgenden Werten im Requestbody:   {„id“:„1“} | | 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 aufrufen mit den folgenden Werten im Requestbody:   {„id“:„1“} | | 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:   {„name“:„Testuser“,  „password“:„admin“} | | HTTP Response: Code 200 |

# Testprotokoll

## Systemtest 1

Getestete Version:

Tester:

Datum, Zeit:

### Testfall 1 „Neues Benutzerkonto erstellen “

Test: ERFÜLLT, NICHT ERFÜLLT

Bemerkung: -

### Testfall 2 „Anmelden“

Test: ERFÜLLT, NICHT ERFÜLLT

Bemerkung: -

### Testfall 3 „Page erstellen“

Test: ERFÜLLT, NICHT ERFÜLLT

Bemerkung: -

### Testfall 4 „Tags hinzufügen“

Test: ERFÜLLT, NICHT ERFÜLLT

Bemerkung: -

### Testfall 5 „Tags bearbeiten“

Test: ERFÜLLT, NICHT ERFÜLLT

Bemerkung: -

### Testfall 6 „Page bearbeiten“

Test: ERFÜLLT, NICHT ERFÜLLT

Bemerkung: -

### Testfall 7 „Page suchen“

Test: ERFÜLLT, NICHT ERFÜLLT

Bemerkung: -

### Testfall 8 „Page löschen“

Test: ERFÜLLT, NICHT ERFÜLLT

Bemerkung: -

### Testfall 9 „Page exportieren“

Test: ERFÜLLT, NICHT ERFÜLLT

Bemerkung: -

### Testfall 10 „Passwort ändern“

Test: ERFÜLLT, NICHT ERFÜLLT

Bemerkung: -

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

Test: ERFÜLLT, NICHT ERFÜLLT

Bemerkung: -

### Testfall 12 „Benutzer reaktivieren“

Test: ERFÜLLT, NICHT ERFÜLLT

Bemerkung: -

# Projektplanung

Siehe Projektplan, Version 3.0 vom 12.12.2017

# Anhang A: Quellcode

*Fügen Sie zuerst eine* ***Übersicht mit allen Dateien*** *(="Inhaltsverzeichnis") des Codes ein. Zum Code gehören auch HTML, CSS, JS, JSP, SQL, etc.*

*Anschliessend folgt der Code.*

***Unbedingt beachten****:*

* *Formatierung und Einrückung*
* *Neue Seite bei neuer Datei*