

Webtechnologien 2 - Projekt-Report
Sommersemester 2018

TU-Do Notes

Eine Web-Anwendung für das Verwalten von nutzerspezifischen ToDo-Notes

Julius Jacobson

Matrikelnr.: 166132

Hatim Bouzian

Matrikelnr.: 198268

Max Walker

Matrikelnr.: 165276

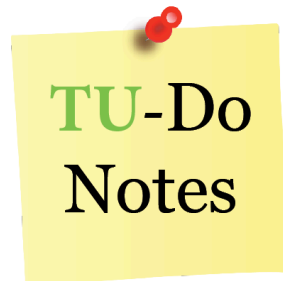
Robin Nowak

Matrikelnr.: 169571

Inhaltsverzeichnis

Einleitung	3
Anwendungsbeispiele	4
Anwendungsbeispiel im privaten Gebrauch	4
Anwendungsbeispiel im gewerblichen Gebrauch	5
Herangehensweise	6
Implementierung	7
Wildfly (JBoss)	7
JPA	8
Probleme	10
GUI Design	12
Fazit der Projektarbeit	12

Einleitung



Den Monitor voll kleben und das Notizbuch vergessen sind Geschichte.

Die Webanwendung TU-Do ist eine digitale Alternative zu den bekannten gelben Klebezetteln. Notizen, Termine und Gedanken können schnell und einfach virtuell erstellt und gespeichert werden. Die Funktion, Notizen auch im Nachhinein bearbeiten, löschen und abhaken zu können, ermöglicht eine übersichtliche Verwaltung. Dank TU-Do gehen Ihnen in Zukunft keine wichtigen Erinnerungen mehr durch die Lappen. Denn mit ihren persönlichen Login-Daten können sie jederzeit und von überall auf Ihre Notizen zugreifen.

Anwendungsbeispiele

Anwendungsbeispiel im privaten Gebrauch

Der erste Schritt um die Webanwendung TU-Do zu nutzen, ist die Account Erstellung / Registrierung. Hierfür muss ein eindeutiger Benutzername und ein Passwort gewählt werden. Ist dies geschehen, kann sofort mit der Erstellung neuer Notizen begonnen werden. Das Eingabefeld „Titel“ sollte mit der Überschrift bzw. einer Zusammenfassung des Notizzettels ausgefüllt werden. Der eigentliche Text des neuen Notizzettels wird in das Eingabefeld „Beschreibung“ eingefügt.

Die Titel der erstellten Todos werden sofort nach der Bestätigung in zeitlich erstellter Reihenfolge angezeigt und können per Mausklick geöffnet, bearbeitet, gelöscht und abgehakt werden. Die Webanwendung TU-Do im privaten Gebrauch hilft den Nutzern Zettelchaos auf dem Schreibtisch zu vermeiden und übernimmt die Aufgaben eines Notizbuches.

The image shows the TU-Do application interface. At the top, the logo 'TU-Do' is displayed in blue. Below it, there are two side-by-side forms. The left form is titled 'Registrieren' and contains two input fields labeled 'Username' and 'Passwort', followed by a 'Registrieren' button. The right form is titled 'Login' and contains two input fields labeled 'Username' and 'Passwort', followed by a 'Login' button.

Abb. 1 - Ausschnitt der grafischen Benutzeroberfläche - Login

Anwendungsbeispiel im gewerblichen Gebrauch

Der Einsatz von TU-Do in Unternehmen bietet nicht nur die normale Notizfunktion wie auch im privaten Gebrauch, sondern dient viel mehr noch als Aufgabenübermittlung, gemeinsame Ideensammlung und als Arbeitskontrolle. Jeder Mitarbeiter des Unternehmens kann auch hier, nachdem ein Benutzername mit Passwort angelegt wurde, eigene Notizen erstellen und verwalten. Der Unterschied zu dem privaten Gebrauch liegt darin, dass der Vorgesetzte der Mitarbeiter Administrator-Rechte in der TU-Do Anwendung besitzt. Dies bietet ihm folgende Möglichkeiten:

- Neue Aufgaben / Termine / etc. können als neue Notiz bei den betroffenen Mitarbeitern hinterlegt werden
- Notizen können gemeinsam beschrieben werden, umso eine Art Brainstorming durchführen zu können, ohne dass die Personen im selben Raum sein müssen
- Es kann kontrolliert werden welche Notizen / Aufgaben von den Mitarbeitern bereits abgehakt wurden. Dies ermöglicht dem Vorgesetzten eine bessere Übersicht, um bei wichtigen Aufgaben sonst gegebenenfalls noch weitere Mitarbeiter auf die bestimmte Aufgabe anzusetzen

Herangehensweise

Der erste Schritt war es, ein Anwendungsfalldiagramm für die TU-Do Anwendung zu erstellen. Dieses zeigt das erwartete Verhalten von dem System bei bestimmten Anwendungen ausgelöst von dem Benutzer. Das Anwendungsfalldiagramm hat uns dabei geholfen, den geforderten Aufgaben aus der Projektaufgabe eine erste Struktur zu geben. Des weiteren wurden bei der Erstellung des Diagramms schon erste Entscheidungen getroffen, wie sich z.B. die Rechte von einem Benutzer zu einem Administrator unterscheiden.

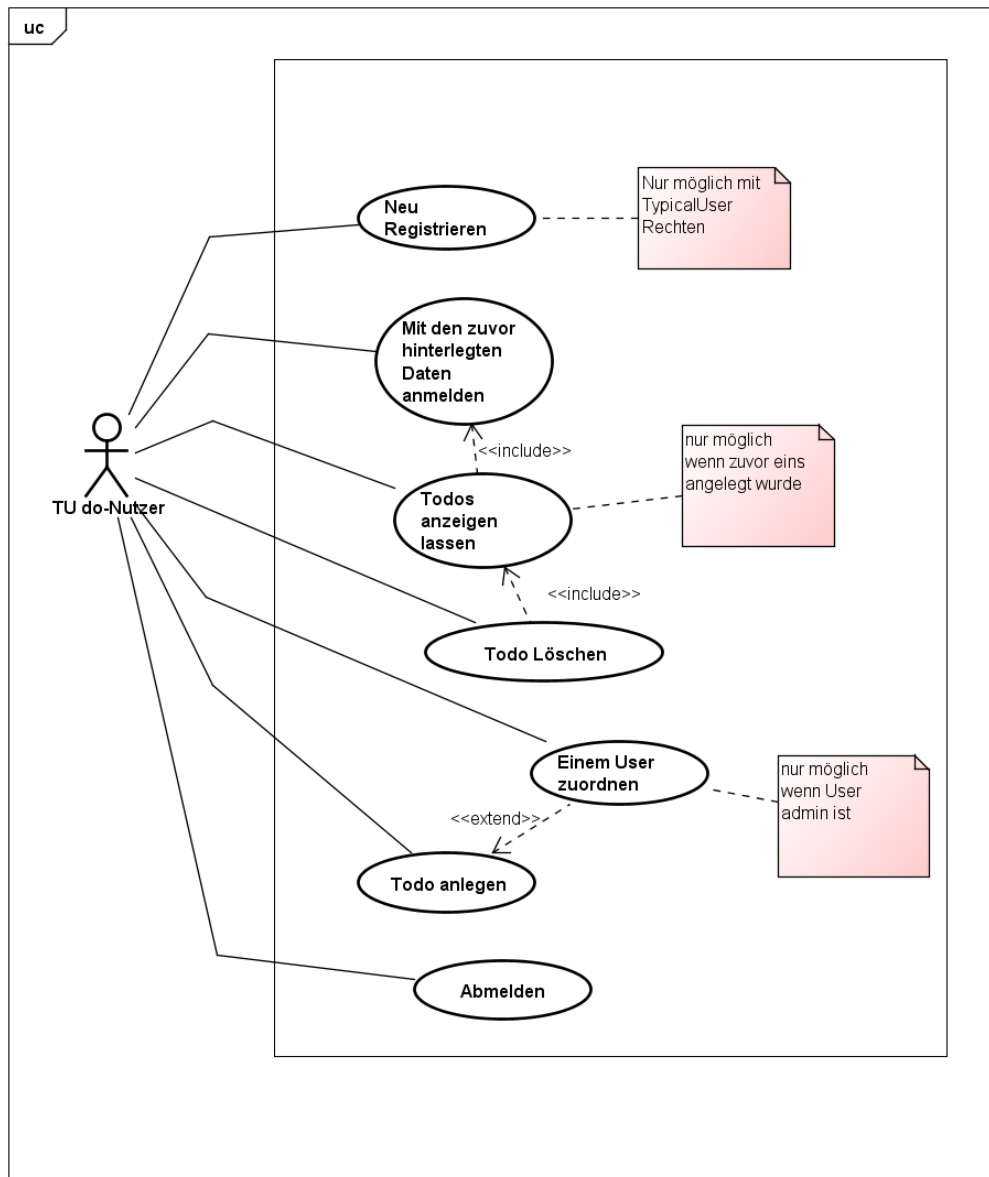


Abb. 2. - Anwendungsfalldiagramm

Implementierung

Wildfly (JBoss)

Zu Beginn des Projektes stellte sich uns die Frage, welchen Anwendungsserver wir implementieren würden. Nachdem sich Tomcat und Wildfly zu den Favoriten herauskristallisiert hatten, verbrachten wir einige Zeit damit Vor- und Nachteile der beiden Möglichkeiten zu betrachten.

Folgende Argumente wurden tabellarisch gegenübergestellt.







Wildfly (JBoss)	Tomcat
 weitaus mehr als nur eine Java Servlet Applikation	 geringer memory footprint
 stellt einen kompletten JEE Stack zur Verfügung	 stellt nicht den gesamten JEE Stack zur Verfügung
 sehr Memorylastig	 schrenkt den Programmierer ein

Abb. 3 - Gegenüberstellung Wildfly - Tomcat

Da wir zu Beginn davon ausgegangen sind, dass wir Zugriff auf den kompletten JEE Stack benötigen würden, haben wir uns für Wildfly entschieden. Außerdem, so zeigte es sich auch in der späteren Implementierung, würde so weniger Aufwand betrieben werden müssen, da wir mit einem Großteil der default Einstellungen arbeiten konnten.

JPA

Um aus unserer Java Anwendung auf die angelegte TU-do Datenbank zugreifen zu können verwenden wir die JPA (Java Persistence API). Vorteil von JPA ist, dass wir mit einem einheitlichen Befehlssatz eine Vielzahl verschiedener DBMS, zum Beispiel Oracle, MySQL, Postgres etc. ansprechen und bearbeiten können.

Um JPA in unserer TU-do Anwendung nutzen zu können, mussten wir erst Abhängigkeiten zu einer JPA-Implementierung in unser Projekt ergänzen. Des Weiteren musste noch die Verbindung zur Datenbank durch das Anpassen der Pfade eingerichtet werden. Diese haben wir mittels Maven pom.xml (Abb. 4) ergänzt.

```
pom.xml x
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>de.ls.wt2</groupId>
8     <artifactId>example08-parent</artifactId>
9     <version>0.1-SNAPSHOT</version>
10    <packaging>pom</packaging>
11
12    <name>WT2 :: ${project.artifactId}</name>
13
14    <properties>
15        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
16
17        <maven.compiler.source>1.8</maven.compiler.source>
18        <maven.compiler.target>1.8</maven.compiler.target>
19
20        <wildfly-swarm.version>2018.4.1</wildfly-swarm.version>
21        <shiro.version>1.4.0</shiro.version>
22
23        <h2.path>${env.HOME}/example07</h2.path>
24        <wildfly.path>D:\Downloads\wildfly-12.0.0.CR1\wildfly-12.0.0.CR1</wildfly.path>
25        <dart-sdk.path>D:\Programs\eclipse\Dart\dart-sdk</dart-sdk.path>
26    </properties>
27
28    <modules>
29        <module>example08-persistence</module>
30        <module>example08-business</module>
31        <module>example08-presentation</module>
32    </modules>
33
34    <dependencies>
35        <dependency>
36            <groupId>javax</groupId>
37            <artifactId>javaee-api</artifactId>
38            <version>7.0</version>
39            <scope>provided</scope>
40        </dependency>
41    </dependencies>
42
43 </project>
44
```

Abb. 4 - Code Ausschnitt: pom.xml

Datenbankkonfiguration

Für die Kommunikation zur Datenbank musste die Verbindung dorthin bekannt gemacht werden. Dies geschieht in der Datei persistence.xml.

```
persistence.xml x
1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.0"
3     xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="
5         http://java.sun.com/xml/ns/persistence
6         http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
7
8     <persistence-unit name="pu">
9         <jta-data-source>java:jboss/datasources/example08</jta-data-source>
10
11         <properties>
12             <!-- configured in example08-ds.xml -->
13             <!--<property name="javax.persistence.jdbc.url" value="jdbc:h2:mem:DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE;"/>-->
14             <!--<property name="javax.persistence.jdbc.url" value="jdbc:h2:file:${env.HOME}/example07" />-->
15             <property name="javax.persistence.jdbc.user" value="sa"/>
16             <property name="javax.persistence.jdbc.password" value=""/>
17
18             <property name="hibernate.hbm2ddl.auto" value="update"/>
19             <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
20
21             <!-- Configure SQL logging -->
22             <property name="hibernate.show_sql" value="true"/>
23             <property name="hibernate.format_sql" value="true"/>
24             <property name="hibernate.use_sql_comments" value="true"/>
25         </properties>
26     </persistence-unit>
27 </persistence>
28
```

Abb. 5 - Code Ausschnitt: persistence.xml

Folgender Codeausschnitt (Abb. 6) zeigt das Erstellen einer neuen digitalen Notiz. Während der Erstellung, werden dem Anwender verschiedene Rechte eingeräumt. Ist er ein Administrator, so kann er die erstellte Notiz auch einem anderen User zuordnen. Nachdem der Anwender Titel, Inhalt und (falls Administrator) auch den passenden User, angegeben hat, wird versucht einen neuen Eintrag in der Datenbank zu erstellen. In diesem Prozess prüft das Programm, ob ein Benutzer angegeben wurde und ob dieser User auch vorhanden ist. Ist dies der Fall, wird der neue Eintrag dem passenden User zugeschrieben und kann nun in der Liste der erstellten Todos angezeigt werden.

```

61
62     @POST
63     @Consumes(MediaType.APPLICATION_JSON)
64     @Produces(MediaType.APPLICATION_JSON)
65     @RequiresAuthentication
66     // @RequiresRoles("admin")
67     public Response create(final DBTodo param) {
68
69         String userName = CRUDHelper.getUserName(SecurityUtils.getSubject());
70         if(userName.equals("")){
71             WT2Realm.WriteDebug("Username is empty, can't create todo");
72             return Response.status(401).build();
73         }
74         DBUser user = CRUDHelper.getUser(entityManager, userName);
75         if(user == null){
76             //Unauthorized
77             WT2Realm.WriteDebug("Can't find user in database, can't create todo");
78             return Response.status(401).build();
79         }
80
81         final DBTodo todo = new DBTodo();
82
83         todo.setTitle(param.getTitle());
84         todo.setDescription(param.getDescription());
85         if(user.getIsAdmin()){
86             todo.setUserName(param.getUserName());
87         }else{
88             todo.setUserName(userName);
89         }
90
91         this.entityManager.persist(todo);
92
93         return Response.ok(todo).build();
94     }
95 }
96

```

Abb. 6 - Code Ausschnitt: TUDO Erstellung

Probleme

1. Session-basierte Authentifizierung:

Es traten Probleme auf die Stelle in dem Beispiel Code zu identifizieren, an welcher geprüft wird ob der Username mit dem Passwort übereinstimmt (Username == Password). An dieser Stelle hätte dann nämlich unser neuer Code hin gemusst, welcher stattdessen überprüft ob der Username in der Datenbank vorhanden ist und ob das dazu gegebene Passwort korrekt ist.

Lösung:

Wir entschieden uns stattdessen für die JWT Authentifizierung, mit der wir unser Problem umgehen konnten. Nachdem der Anwender seine Login Daten und das Passwort in dem Anmeldebereich der Webapplikation eingegeben hat, werden diese zur Erzeugung eines JWT-Tokens an die AuthenticationRest Klasse weitergegeben. Zunächst wird überprüft, ob es in der Datenbank einen User gibt, der mit dem übergebenen Namen übereinstimmt. Ist dies der Fall, folgt die Überprüfung des

Passworts. Werden diese beiden Überprüfungen ohne Fehlermeldung ausgeführt, wird nun ein Token generiert und der Anwender kommt in seinen persönlichen Notizzettel Bereich.

```
AuthenticationREST.java x
20 createJWT()
21 @Transactional
22 public class AuthenticationREST {
23
24     @PersistenceContext
25     private EntityManager entityManager;
26
27     @Path("authenticate")
28     @POST
29     @Consumes(MediaType.APPLICATION_JSON)
30     public Response createJWTToken(JWTLoginData credentials) throws JOSEException {
31
32         // do some proper lookup
33         final String user = credentials.getUsername();
34         final String pwd = credentials.getPassword();
35
36         WT2Realm.WriteDebug("Posted auth info: user="+user+", pw="+pwd);
37
38         DBUser dbUser = CRUDHelper.getUser(entityManager, user);
39
40         if(dbUser == null){
41             return Response.status(Status.UNAUTHORIZED).build();
42         }
43
44         if (!dbUser.getPassword().equals(pwd)) {
45             WT2Realm.WriteDebug("Retrieved from db: user="+dbUser.getUserName()+" pw="+dbUser.getPassword());
46             return Response.status(Status.UNAUTHORIZED).build();
47         }
48         String token = JWTUtil.createJWTToken(credentials);
49         WT2Realm.WriteDebug("Generated Token "+token);
50         return Response.ok(token).build();
51     }
52 }
```

Abb. 6 - Code Ausschnitt: AuthenticationREST

2. JWT- „logout“:

Der eingebaute JWT Logout hat lediglich clientseitig den Token gelöscht und hat die Komponente so geändert, dass statt „logout“, „einloggen..“ Angezeigt wird.

Serverseitig konnte die Session trotz Logout noch identifiziert werden (vermutlich mittels Cookie JSESSIONID). Das Ummelden, ohne vorher die Cookies zu löschen oder einen neuen Browser zu verwenden, war deshalb nicht möglich.

Lösung:

Bei einem Klick auf „logout“ , eine http GET-Request an /logout senden und für diesen Endpunkt eine neue Methode zu Exposee, in der die Session vom aktuellen User gecleared wird.

GUI Design

Der große Vorteil der reellen Notizzetteln gegenüber jeglicher digitaler Version, ist die Einfachheit und die Geschwindigkeit in der sie erstellt werden können. Dies ist der Grund warum wir unser Interface so einfach und übersichtlich wie nur Möglich gestaltet haben. Durch die Funktion „Mit den zuvor hinterlegten Daten anmelden“ ist es nicht mal notwendig den Benutzernamen und das Passwort bei jeden Login erneut einzugeben. Diese Funktion und das einfache und verständliche Design der grafischen Benutzeroberfläche, machen die TU-Do Anwendung zu der perfekten Notizzettel Alternative.

Fazit der Projektarbeit

Die Schwierigkeit der Aufgabe war genau richtig gewählt um die in der Vorlesung vorgestellten Konzepte anzuwenden und zu verinnerlichen.

Unsere Gruppe bestand zunächst aus zwei Teilgruppen und in beiden Teilen lief die Bearbeitung der Aufgabe eher schleppend.

Nach der Zusammenschließung und einer extrem guten Kommunikation und Arbeitsaufteilung nahm die Bearbeitung endlich Fahrt auf.

Verglichen mit wöchentlichen Abgaben, war es schwieriger einen Start in die Bearbeitung der Projektaufgabe hinzubekommen, da man sehr viel Zeit zur Verfügung hat und man die Aufgabe von der einen zur nächsten Woche verschiebt. Im Nachhinein hätten wir das Angebot der Hilfestunde am Freitag früher wahrnehmen sollen, damit das Grundgerüst der Anwendung früher gestanden hätte.