# Kanban-Board à la Trello

Im Modul *Workshop Distributed Software Systems* werden die Themen der Vertiefungsrichtung *Verteilte Software Systeme* in einem durchgehenden Beispiel angewendet und umgesetzt. Dieses Dokument beschreibt die Aufgabenstellung für die Durchführung 2016.

## 1. Problembeschreibung

In diesem Jahr soll im Rahmen des Moduls DSS Workshop eine Plattform entwickelt werden, mit der Kanban-Tafeln abgebildet werden können (https://de.wikipedia.org/wiki/Kanban-Tafel), ähnlich wie das in Trello realisiert worden ist (http://trello.com). Auf einer Kanban-Tafel werden Arbeitselemente (Tasks) in Kolonnen dargestellt, z.B. „To Do“, „Doing“ und „Done“. Benutzer sollen auf der Plattform neue Projekte erstellen können, auf denen dann neue Tasks erfasst, bearbeitet und zwischen den verschiedenen Kolonnen hin und her verschoben werden können.

## 2. Zielsetzung

Das Ziel dieser Arbeit liegt darin, ein relativ einfaches Geschäftsmodell mit Hilfe von Internet-Technologien komponentenbasiert abzubilden. Wir wollen uns dabei auf die im Unterricht vorgestellten Plattformen konzentrieren, also Spring oder JEE. Für das GUI verwenden Sie einen Web-Container (Servlets; JSP, JSF oder RIA mit REST) und für die Geschäftslogik Spring (mit Spring-Data) oder einen EJB-Container (CDI, Session-Beans, JPA).

Da die Aufgabe ziemlich umfangreich ist, soll sie in Gruppen zu 3-4 Studierenden bearbeitet werden. Die Arbeit wird benotet und ergibt die Modulnote.

## 3. Anforderungsbeschreibung

### Funktionale Anforderungen

Ein Projekt auf der zu entwickelnden Plattform entspricht einem Board, welches Task-Listen enthält. Jede Task-Liste besteht aus einer Menge von „Karten“, die untereinander angeordnet sind. Diese Karten entsprechen je einem zu erledigenden Task. Für unser Projekt sollen fix die drei Listen TODO, DOING und DONE vorgegeben sein.

Jeder Task hat einen Titel, optional eine verantwortliche Person und eine optionale Beschreibung. Zu einem Task sollen auch mehrere Anhänge (z.B. Bilder, Dokumente etc.) hinzugefügt und auch wieder gelöscht werden können.

Filter sollen verwendet werden können, um die Menge der Tasks einzuschränken und zwar nach verantwortlicher Person oder nach Stichworten.

Ein Projekt kann von einem Benutzer erstellt werden. Auf einem Projekt arbeiten jedoch typischerweise mehrere Benutzer. Der Besitzer eines Projektes kann weitere Benutzer zu einem Projekt einladen, die dann dieses Projekt ebenfalls bearbeiten können, d.h. die neue Tasks erstellen können und diese auch verschieben können.

Im Gegensatz zu Trello soll es möglich sein, alle Tasks, die einem zugeordnet sind, auf einer virtuellen Seite mit den vorgegebenen Listen TODO, DOING und DONE darzustellen. Optional soll es auch möglich sein, die Tasks aller Projekte, für die man verantwortlich ist, auf einer virtuellen Seite mit den vorgegebenen Listen TODO, DOING und DONE darzustellen (und dann darauf auch zu filtern).

Um auf der Plattform aktiv zu werden, muss sich ein Benutzer anmelden. Neue Benutzer können sich selbständig auf der Plattform registrieren.

Ihre Plattform soll regelmässig Notifikationen an die Nutzer verschicken (was genau für Notifikationen verschickt werden ist Ihnen überlassen). Für Projekt-Owner könnten die Notifikationen aus wöchentlichen E-Mails zum Projekt-Stand (x TODO, y DOING, z DONE) bestehen. Projekt-Teilnehmer könnten

täglich per E-Mail informiert werden welche neuen Tasks ihnen zugeordnet worden sind. Sehr Interessant wären auch Notifikationen welche Deltas angeben (z.B. „diese Woche wurden xyz Tasks erledigt“).

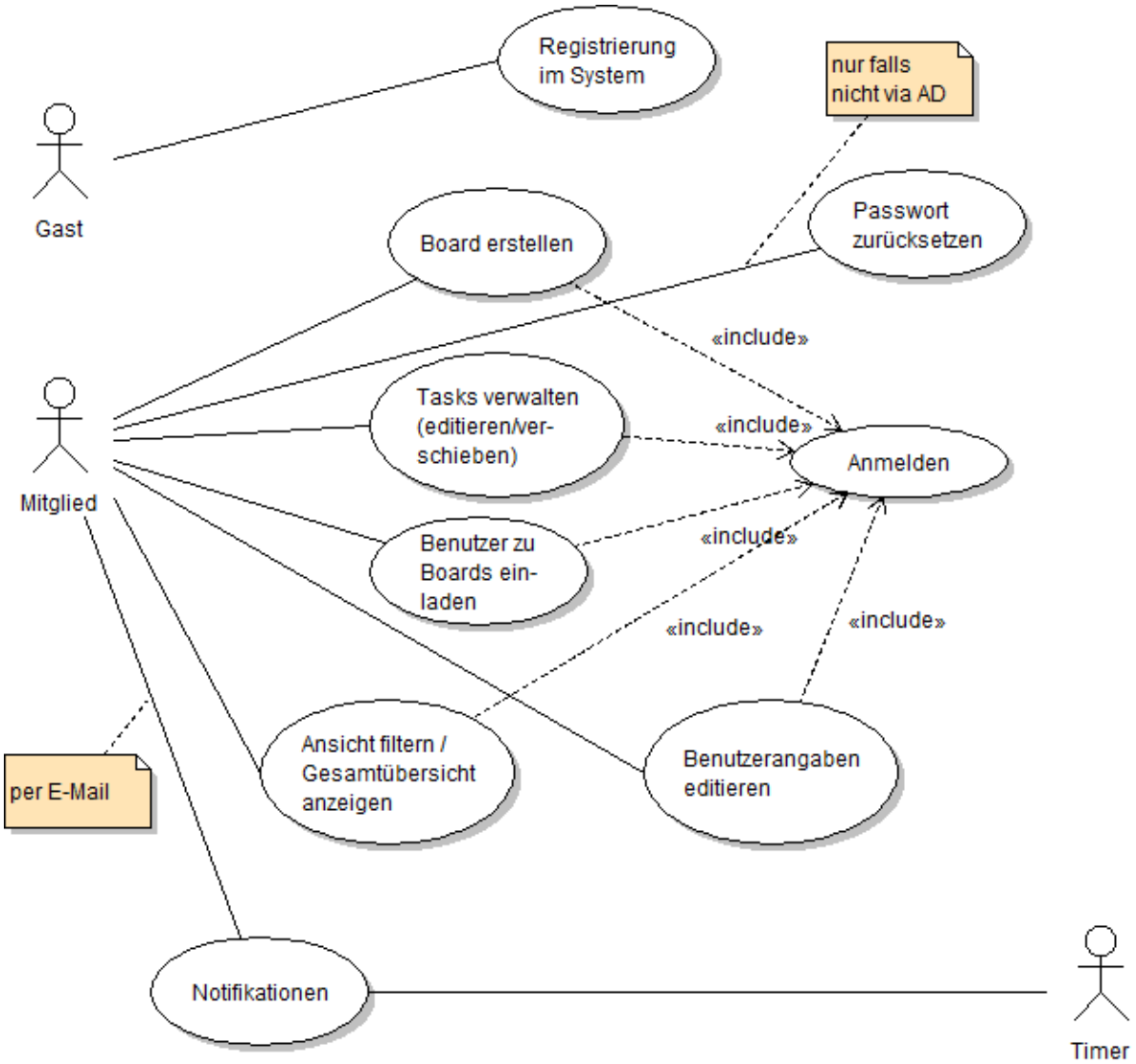


*Abbildung 1: Use Case Diagramm*

Folgende Rollen sind vorgesehen:

*Gast*
Kann sich auf der Plattform registrieren. Seine Email-Adresse soll dabei validiert werden.

*Mitglied*
Benutzer der Projektplattform. Er kann neue Boards erstellen und auch weitere Benutzer zu seinen Boards einladen. Auf einem Board kann er neue Tasks erstellen und bestehende editieren, verschieben oder löschen. Auf einem Board kann er die angezeigten Tasks einschränken, entweder nach Stichwort oder nach verantwortlicher Person. Zudem kann er ein virtuelles Board anzeigen, das aus all seinen Tasks besteht.

*Timer*
Ein Timer initiiert den Versand von Notifikationen.

**Nichtfunktionale Anforderungen**

Technologie:
- Die Applikation muss sowohl auf Unix-, Mac-OS- und auch auf Windows-basierten Systemen eingesetzt werden können.
- Es müssen die modernen Browser-Versionen unterstützt werden (Chrome > 48, Firefox > 38).
- Es soll ein *Java Application Server* verwendet werden (Tomcat oder JavaEE)

Benutzerinterface:
- Die Applikation soll ohne Benutzerhandbuch bedienbar sein.
- Die Sprache der Applikation ist Deutsch. Eine Unterstützung für die Internationalisierung der Webpage soll aber vorhanden sein.

Architektur:
- Eine saubere Schichtenarchitektur ist notwendig. Es soll möglich sein, die Serviceschicht und die Serverapplikation später auch über weitere Clients nutzen zu können (z.B. mobile Applikation), ohne dass Änderungen im Businesslayer erforderlich werden.
- Alle relevanten Konfigurationseinstellungen müssen ohne Kompilation der Applikation möglich sein und bei einem Neustart des Servers aktiv werden.

Sicherheit:
- allfällige Passwörter sind weder im Klartext noch (reversibel) verschlüsselt abgelegt
    - https://owasp.org/index.php/Password_Storage_Cheat_Sheet
    - https://github.com/Squeng/Apsi/blob/4027e82bf754ca206a3cd49becb8f46a48f17d32/app/value_types/Password.java (als Beispiel)
    - https://owasp.org/index.php/Forgot_Password_Cheat_Sheet
- das Zugriffskontrollmodell ist adäquat
    - https://owasp.org/index.php/Access_Control_Cheat_Sheet
- die App weist keine SQL-Injection-Schwachstellen auf
    - https://owasp.org/index.php/Query_Parameterization_Cheat_Sheet
    - https://owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
- die Kommunikation erfolgt einzig über HTTPS
    - entsprechende HTTP-Response-Headers werden genutzt
    - Falls die Web-App gegenüber einem Web-Service auch als Client fungiert, siehe http://www.cs.utexas.edu/~shmat/shmat_ccs12.pdf
- weitere HTTP-Response-Headers werden sinnvoll genutzt
    - for Servlet-based apps: https://github.com/sourceclear/headlines
- die App weist keine XSS-Schwachstelle auf
    - https://owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
- die App weist keine CSRF-Schwachstelle auf
    - Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet, https://owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

siehe auch:
- Iron-Clad Java: Building Secure Web Applications http://www.mhprofessional.com/product.php?isbn=0071835881&cat=112
- http://blog.squeng.com/2016/01/24/apsi/

## 4. Architektur und Aufgaben

Grundsätzlich soll diese Applikation in mehreren logischen Schichten aufgebaut werden:

- Präsentationsschicht
- Geschäftslogikschicht
- Datenbankschicht

Die Datenbankschicht bildet eine standardisierte Schnittstelle zur Datenbank. Das zugrunde liegende Datenbankmodell wird durch diese Schicht verborgen. Diese Schicht ist in Java mit JPA zu realisieren.

Basierend auf der Datenbankschicht baut die Geschäftslogikschicht auf. In der Geschäftslogik werden die Geschäftstransaktionen abgewickelt. Eine Geschäftstransaktion bildet die Geschäftsregeln ab (z.B. das Verschieben eines Tasks). Die Geschäftslogik soll mit EJB Session Beans oder auf Basis des Spring Frameworks realisiert werden. Zur Laufzeit soll ein aussagekräftiges Logging die Nutzung der Logik aufzeigen.

Die Präsentationsschicht muss die Daten mit einem Webinterface zur Verfügung stellen. Für die Generierung der HTML Seiten kann JSP oder ein MVC Model2 Framework (wie JSF oder Spring MVC) oder ein Ajax-basiertes Framework (z.B. AngualrJS mit REST) verwendet werden.

Für den Versand von Emails (Emailbestätigung, Password-Reset) verwenden Sie das JavaMail API.

Es muss mindestens ein Batch-Job vorhanden sein, der im Background eine Aufgabe in einem regelmässigen Intervall ausführt. Die Aufgabe selber kann vom Projektteam definiert werden.

Für die Authentifizierung / Autorisierung auf Ihrer Plattform ist JAAS oder Spring Security zu verwenden.

Für Ihre Geschäftslogik sollen Test-Fälle bereitgestellt werden (JUnit-Tests). Ihr Projekt soll am Ende so bereitgestellt sein, dass wir Ihre Projekte lokal installieren und die Testfälle ausführen können.

## 5. Organisation

Da die Aufgabe ziemlich umfangreich ist, soll sie in Gruppen zu 3-4 Studierenden bearbeitet werden (drei Gruppen mit vier, eine mit drei Studierenden). Die Applikation und die Dokumentation werden benotet und ergibt die Modulnote.

## 6. Termine

Mo, 22.02.16 Formierung der Teams (Angabe per email)
15 Studierende => typischerweise 3er Teams

Mo, 07.03.16 Abgabe eines Projektplans (per Email an die Betreuer)
Angaben zu Grobarchitektur und Rollenzuteilung sowie ein ungefährer Zeitplan

Mo, 04.04.16 Abgabe folgender Dokumente:
- GUI Vorschlag (auch als Papierprototyp möglich, der Ablauf der Webapplikation soll daraus ersichtlich sein).
- Statisches Klassendiagramm (Datenbankschema)
- Beschreibung der Schnittstellen der Business-Logik. Hier neben Syntax auch Semantik angeben!

Mo, 11.04.16 Zwischenreview, für alle Studierenden
Kurze Präsentation der zu realisierenden Lösung und anschliessende Diskussion mit den Dozierenden.

Mo, 30.05.16 Abgabe der Lösung: Diese enthält folgende Teile:
- Dokumentation (Beschreibung der Lösung)
- Code auf CD (inkl. Build Umgebung und allen eingesetzten Bibliotheken)
- URL auf lauffähige Version

Mo, 06.06.16 Präsentation der Lösungen (für alle Studierenden)

Mo, 13.06.16 Verteidigungen / Projektklärungen, gruppenweise (15-19 Uhr)

*Dokumentation*
Als Dokumentation erwarten wir eine Beschreibung Ihrer Lösung. Es sollten alle wichtigen Designentscheidungen, die Sie getroffen haben, beschrieben sein. In der Dokumentation sollen auch interessante Aspekte und Lösungen dargestellt werden und die Dokumentation soll helfen, dass wir uns im Code zurechtzufinden. Ein UML-Diagramm Ihrer Klassen ist dazu geeignet (bzw. zwingend nötig).

Den Code geben Sie bitte auf CD oder Memory-Stick ab. Das Projekt soll von uns einfach installiert und getestet werden können (durch Ausführung der Testfälle).

Ihre Lösung soll auch auf einem Rechner zugreifbar sein, damit wir Ihre Lösung (mind. im FHNW Intranet) anschauen können. Geben Sie im Bericht die URL der Startseite an.

Geben Sie im Bericht auch an, wer in Ihrem Team welche Teile realisiert hat.

*Präsentation*
In der Präsentation soll auf jene Aspekte hingewiesen werden, welche die anderen Gruppen interessieren könnten, z.B. spezielle Probleme, elegante Lösungen (wie Authentisierung/Autorisierung, GUI Framework, .....), verwendete Technologien, Erfahrungen, etc.
Die Präsentation soll auch eine Live-Demo des Systems enthalten.

Mögliche Themen der Präsentation sind also:
- Architektur Ihrer Lösung
- Erfahrungen mit Web Framework
- Erfahrungen mit Applikationsserver
- Spezielle Sicherheitsmerkmale, die Sie in Ihrer Applikation berücksichtigt haben

Sie haben für diese Präsentation 20 Min Zeit (inkl. Diskussion)

## 7. Bewertung

Ihre Arbeit (Dokumentation + Code) wird mit einer Projektnote bewertet. In diese Note fliessen folgende Aspekte ein:

- Fachliche Umsetzung: 65%
- Dokumentation: 20%
- Präsentation: 15%

Pro Gruppe gibt es im Normalfall *eine* Projektnote.

Nach der Abgabe der Dokumentation findet eine Klärung statt. In dieser Klärung werden die Dozierenden dem Team spezifische Fragen zur Lösung stellen. Fragen zu allen Themengebieten können an alle gerichtet werden. Diese Klärung dauert 60 Min. Anwesend sind alle am Modul beteiligten Dozierenden und die Mitglieder jedes Teams.

Die Projektnote kann auf Grund der Klärung individuell korrigiert werden.

Der Studierende hat das Modul bestanden, falls

- die individuelle Projektnote genügend ist (>= 3.8)
- die Testat-Bedingung erfüllt ist, d.h. Anwesenheit bei Zwischenreview, Schlusspräsentation und Verteidigung.

## 8. Betreuung

Die Dozierenden stehen bei Fragen zur Verfügung:

| | |
|---|---|
| - Sicherheit: | Paul Sevinç |
| - Web-Framework / Architektur: | Jürg Luthiger |
| - Services / ORM | Dominik Gruntz |

Wir empfehlen bei Problemen einen Termin zu vereinbaren und vorab das Problem per Email zu schildern.