1 Webanwendung "Bug-Tracker"

1.1 Beschreibung der Anwendung

"Bug-Tracker" sind ein Hilfsmittel bei der Qualitätssicherung von Software-Entwicklungsprojekten: erkannte Mängel werden protokolliert und weitere Bearbeitungsvorgänge zur Beseitigung der Mängel angestoßen.

Zur Vereinfachung der Aufgabenstellung wird hier nur der folgende Funktionsumfang berücksichtigt:

- erkannte Fehler werden dokumentiert; dabei werden erfasst:
  - Beschreibung des Fehlers
  - der Bearbeiter der Fehlererfassung (QS-Mitarbeiter)
  - das Datum der Erfassung
  - die Fehler werden Komponenten, die in Projekten entwickelt werden, zugeordnet
  - die Fehler werden Kategorien zugeordnet
- die Beseitigung von Fehlern wird mit einer Beschreibung erfasst, zusätzlich werden:
  - der Bearbeiter der Fehlerbeseitigung (SW-Entwickler) vermerkt
  - das Datum der Beseitigung dokumentiert
  - die Fehlerursache beschrieben und einer Kategorie zugeordnet
- zur Zuordnung der Fehler werden Projekte und die darin vorgesehenen Komponenten verwaltet
- es werden Kategorien für die Fehler und die Fehlerursachen verwaltet
- es werden zwei Auswertungen vorgesehen:
  - Zusammenstellung der Fehler nach Projekt / Komponente / Status Fehler (erkannt/beseitigt)
  - Zusammenstellung der Fehler nach Kategorie / Status Fehler (erkannt/beseitigt).

1.2 Zustandsmodell

Fehler gibt es also in 2 Zuständen:

- erkannt
- beseitigt.

Bei der Bearbeitung der Fehler werden verschiedene Systemzustände durchlaufen:

- "Fehler ist protokolliert": wird erreicht durch das Ereignis "Fehler erkannt", als Aktion wird "Fehler erfassen" durchgeführt
- "Entwickler zugewiesen": wird erreicht durch das Ereignis "Entwickler zuweisen", womit die Beseitigung des Fehlers einem Entwickler zugewiesen wird
- "Fehler behoben": wird erreicht durch das Ereignis "Bearbeitung erfolgt", als Aktion bei diesem Zustandsübergang wird "Fehler beheben" durchgeführt
- "Geprüft": wird erreicht durch die Meldung des Mitarbeiters; dabei wird die Aktion "Fehlerbehebung prüfen" durchgeführt.

Die beiden folgenden Zustandsübergänge sind abhängig vom Ergebnis der Prüfung. Dazu werden im Zustandsdiagramm 2 Bedingungen angegeben:

- "[Prüfung erfolgreich]": wenn die Bedingung zutrifft, wird der Endzustand erreicht, dabei wird die Aktion "Freigeben" durchgeführt
- "[Prüfung nicht erfolgreich]": wenn die Bedingung zutrifft, wird wieder der Zustand "Fehler ist protokolliert" erreicht, dabei wird die Aktion "Fehler erfassen" durchgeführt; das Ergebnis der gescheiterten Prüfung wird als neuer Fehler erfasst.

Diese Abfolge ist in der folgenden Abbildung dargestellt.

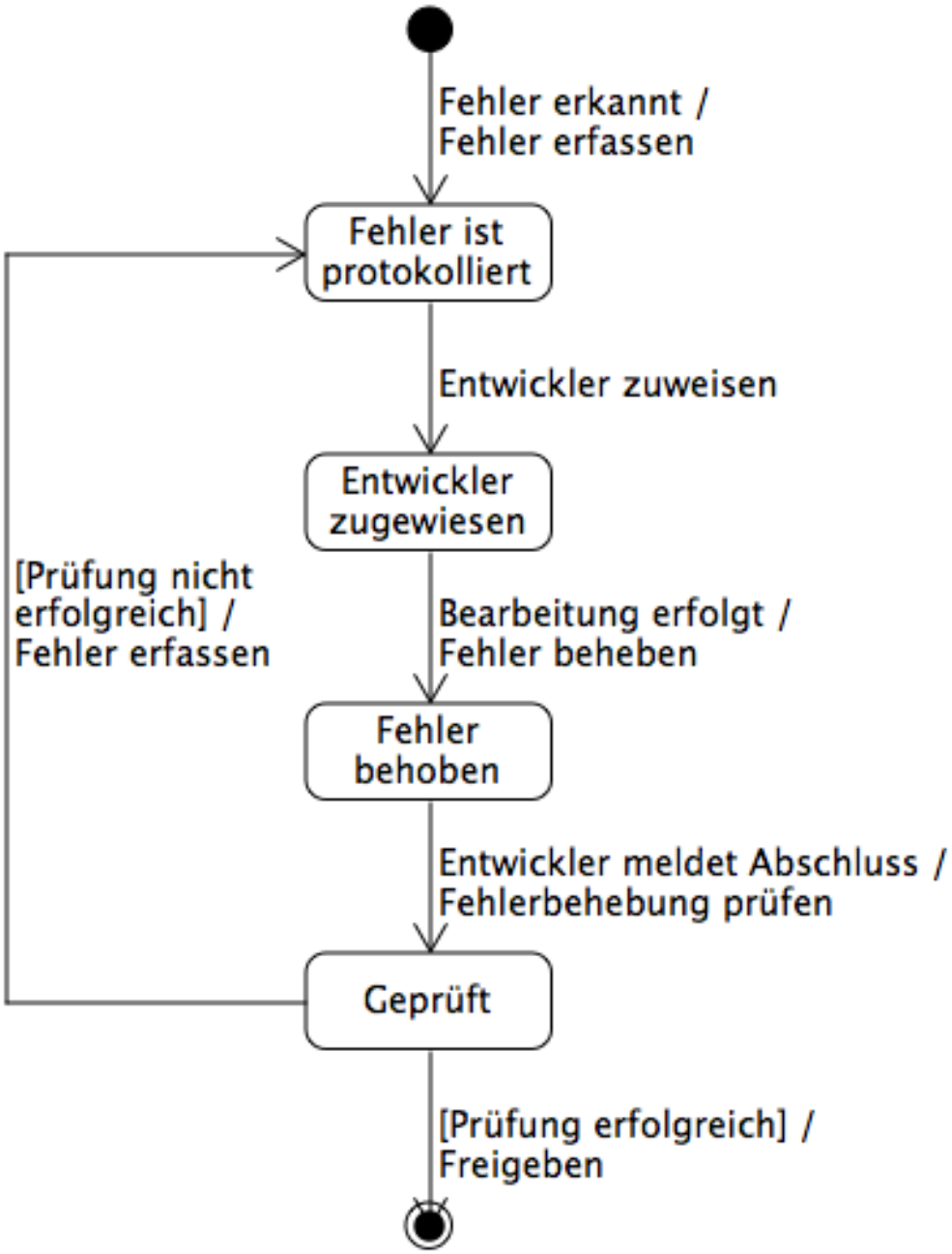

Abbildung 1: Zustandsmodell Bug-Tracker

1.3 Datenmodell

Das Datenmodell der Webanwendung sieht entsprechend den obigen Angaben so aus:

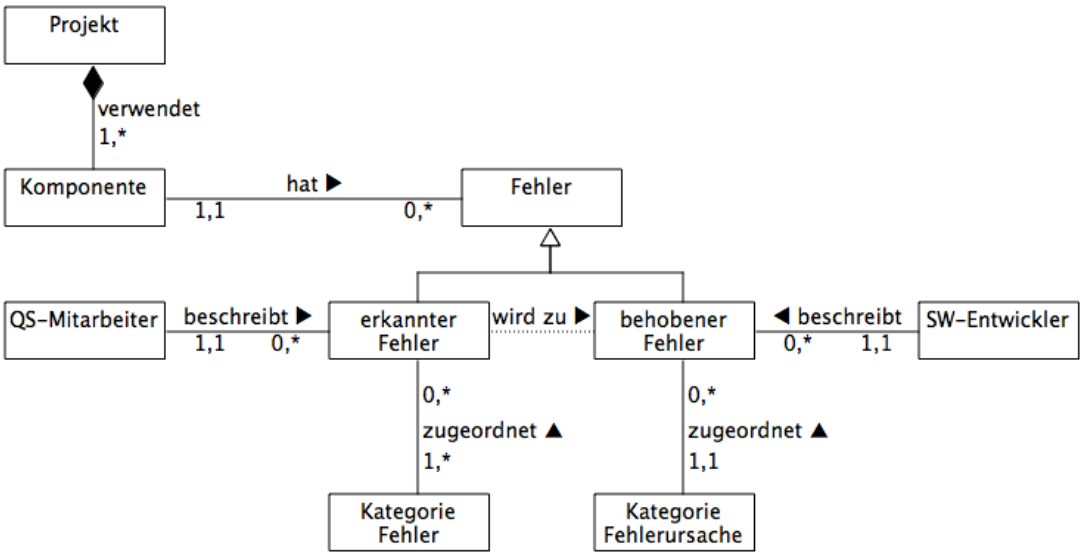

Abbildung 2: Datenmodell Bug-Tracker

Beachten Sie:

- Fehler beziehen sich eindeutig auf eine Komponente
- erkannte Fehler werden durch genau einen QS-Mitarbeiter beschrieben
- erkannten Fehlern wird mindestens eine Fehlerkategorie zugeordnet, ggf. werden auch mehrere zugeordnet
- behobene Fehler werden durch genau einen SW-Entwickler beschrieben
- behobenen Fehlern wird genau eine Fehlerursachenkategorie zugewiesen.

2 Anforderungen an die Umsetzung

Erstellen Sie die Webanwendung "Bug-Tracker" als Client-Server-Anwendung, die die Architekturpinzipien *REST* (*Representational State Transfer*) und *Single-Page-Application* berücksichtigt:

- es wird einmalig das Dokument `index.html` als Webseite geladen
- weitere Inhalte werden dynamisch in Container, die im Markup des Dokuments `index.html` vorhanden sind, geladen
- alle notwendigen Ressourcen - CSS-Stilregeln in externen CSS-Dateien, javascript-Code in externen Quelldateien - werden durch das Dokument `index.html` referenziert und damit einmalig mit dem Laden des Dokuments geladen
- zur Kommunikation zwischen Client und Server wird der *AJAX*-Ansatz verwendet:
  - Anforderungen, ggf. mit Daten, werden im Hintergrund asynchron an den Server gesendet:
    - gemäß den Prinzipien des REST-Ansatzes muss hierbei die HTTP-Methode angegeben werden
    - Daten werden im Message-Body als Key-Value-Paare übertragen (Standard, wie dies bei einfachen Formularen mit der Methode POST erfolgt)
  - Daten, die vom Server an den Client übertragen werden, werden grundsätzlich gemäß der *JSON*-Konvention strukturiert.

Die Aufbereitung der Daten und die Erzeugung des Markups erfolgt auf dem Client mit der Template-Engine `te/tm`. Die Templates selber werden als statische Ressource vom Server geliefert.

Beachten Sie die Hinweise im Anhang (Anwendungsbeispiel).

Method-Dispatching
Serverseitig wird *Method-Dispatching* verwendet: die HTTP-Methode entscheidet, welche Methode aufgerufen wird. In der Dokumentation zu `cherrypy` wird im Tutorial 7 gezeigt, wie man das *Method-Dispatching* verwendet (siehe http://docs.cherrypy.org/en/latest/tutorials.html?highlight=dispatching#tutorial-7-give-us-a-rest). Es ist **nicht** erforderlich, die im Abschnitt "Advanced" der

Dokumentation zu `cherrypy` beschriebene Vorgehensweise "RESTFul-style dispatching" zur Auswertung komplexerer Pfade in den URI anzuwenden.

2.1 Vereinfachung

Zur Vereinfachung ist eine Unterscheidung der beiden Rollen "QS-Mitarbeiter" / "SW-Entwickler" nicht in Form unterschiedlicher Anmeldungen erforderlich:

- eine Benutzeranmeldung ist nicht erforderlich
- "QS-Mitarbeiter" und "SW-Entwickler" werden in zwei Datenbeständen geführt, so dass eine Zuordnung zu den erkannten bzw. behobenen Fehlern möglich ist.

2.2 Beschreibung der Anfragen

Es sollten etwa die nachfolgend beschriebenen Anfragen berücksichtigt werden (Sie müssen diese Liste überprüfen und ggf. erweitern / anpassen [ohne die Systematik zu ändern!])

Methode	**URI**	**Reaktion**
GET	/projekt/	alle Projekte (Liste) anfordern
GET	/projekt/:projekt-id	einzelnes Projekt gemäß :projekt-id mit Liste der Komponenten anfordern
PUT	/projekt/ + Daten	neues Projekt speichern, :projekt-id zurückgeben
POST	/projekt/:projekt-id + Daten	Projekt mit der Id :projekt-id ändern
DELETE	/projekt/:projekt-id	Projekt mit der Id :projekt-id löschen (incl. aller abhängigen Daten)

Methode	**URI**	**Reaktion**
GET	/projektkomponenten/:projekt-id	Komponenten zum Projekt gemäß :projekt-id (Liste) anfordern
GET	/komponente/	alle Komponenten (Liste, mit Verweis auf das zugeordnete Projekt) anfordern
GET	/komponente/:komponente-id	einzelne Komponente gemäß :komponente-id anfordern
PUT	/komponente/:projekt-id + Daten	neue Komponente zum Projekt gemäß :projekt-id speichern, :komponente-id zurückgeben
POST	/komponente/:komponente-id + Daten	Komponente mit der Id :komponente-id ändern
DELETE	/komponente/:komponente-id	Komponente mit der Id :komponente-id löschen (incl. aller abhängigen Daten)

Methode	**URI**	**Reaktion**
GET	/qsmitarbeiter/	Daten aller QS-Mitarbeiter (Liste) anfordern
GET	/qsmitarbeiter/:qsmitarbeiter-id	Daten eines einzelnen QS-Mitarbeiters gemäß :qsmitarbeiter-id anfordern
PUT	/qsmitarbeiter/ + Daten	Daten eines neue QS-Mitarbeiters speichern, :qsmitarbeiter-id zurückgeben
POST	/qsmitarbeiter/:qsmitarbeiter-id + Daten	Daten eines QS-Mitarbeiters mit der Id :qsmitarbeiter-id ändern
DELETE	/qsmitarbeiter/:qsmitarbeiter-id	Daten eines QS-Mitarbeiter mit der Id :qsmitarbeiter-id löschen (incl. aller abhängigen Daten)

Methode	**URI**	**Reaktion**
GET	/swentwickler/	Daten aller SW-Entwickler (Liste) anfordern
GET	/swentwickler/:swentwickler-id	Daten eines einzelnen SW-Entwicklers gemäß :swentwickler-id anfordern
PUT	/swentwickler/ + Daten	Daten eines neue SW-Entwicklers speichern, :swentwickler-id zurückgeben
POST	/swentwickler/:swentwickler-id + Daten	Daten eines SW-Entwicklers mit der Id :swentwickler-id ändern
DELETE	/swentwickler/:swentwickler-id	Daten eines SW-Entwickler mit der Id :swentwickler-id löschen (incl. aller abhängigen Daten)

Methode	**URI**	**Reaktion**
GET	/katfehler/	alle Fehlerkategorien (Liste) anfordern
GET	/katfehler/:katfehler-id	einzelne Fehlerkategorie gemäß :katfehler-id anfordern
PUT	/katfehler/ + Daten	neue Fehlerkategorie speichern, :katfehler-id zurückgeben
POST	/katfehler/:katfehler-id + Daten	Fehlerkategorie mit der Id :katfehler-id ändern
DELETE	/katfehler/:katfehler-id	Fehlerkategorie mit der Id :katfehler-id löschen

Methode	**URI**	**Reaktion**
GET	/katursache/	alle Fehlerursachenkategorien (Liste) anfordern
GET	/katursache/:katursache-id	einzelne Fehlerursachenkategorie gemäß :katursache-id anfordern
PUT	/katursache/ + Daten	neue Fehlerursachenkategorie speichern, :katursache-id zurückgeben
POST	/katursache/:katursache-id + Daten	Fehlerursachenkategorie mit der Id :katursache-id ändern
DELETE	/katursache/:katursache-id	Fehlerursachenkategorie mit der Id :katursache-id löschen

Methode	**URI**	**Reaktion**
GET	/fehler/	alle Fehler (Liste) (mit Verweisen auf Komponenten/Projekte, QS-M. oder SW-Entw.) anfordern
GET	/fehler/?type=erkannt	alle erkannten Fehler (Liste) (mit Verweisen auf Komponenten/Projekte, QS-M. oder SW-Entw.) anfordern
GET	/fehler/?type=behoben	alle behobenen Fehler (Liste) (mit Verweisen auf Komponenten/Projekte, QS-M. oder SW-Entw.) anfordern
GET	/fehler/:fehler-id	einzelnen Fehler gemäß :fehler-id anfordern
PUT	/fehler/ + Daten	neuen Fehler speichern, :fehler-id zurückgeben
POST	/fehler/:fehler-id + Daten	Fehler mit der Id :fehler-id ändern

Die beiden Fehlerarten (Unterklassen "erkannter Fehler" und "behobener Fehler") können durch das Attribut "type" implementiert werden. Bei der Erfassung eines neuen Fehlers wird immer der Typ "erkannt" eingetragen. Die Änderung zu einem behobenen Fehler erfolgt mit einer `PUT`-Operation. Das direkte Löschen von Fehlern wird nicht vorgesehen.

Die beiden geforderten Auswertungen sind ebenfalls Ressourcen, für die es jedoch nur `GET`-Anfragen gibt:

Methode	**URI**	**Reaktion**
GET	/prolist/	Auswertung Fehler nach Projekt/Komponente/Status als Liste anfordern
GET	/katlist/	Auswertung Fehler nach Kategorie/Status als Liste anfordern

Die clientseitig zu verwendenden Templates werden mit folgender Anforderung geladen:

Methode	**URI**	**Reaktion**
GET	/templates/	Templates (JSON-Format, siehe tm.js, Key = 'templates')

Mit Doppelpunkt gekennzeichnete Elemente sind variable Werte. Die Daten bei PUT- und POST-Anfragen werden im Message-Body übertragen.

Die Ergebnisse der Anfragen werden stets im JSON-Format zurückgeliefert. Der HTTP-Statuscode wird verwendet, um auf korrekt durchgeführte oder nicht ausführbare Anfragen hinzuweisen (z.B. 200 = Ok, 404 = angefragte Daten sind nicht vorhanden).

Beachten Sie die Hinweise im Anhang.

2.3 Benutzungsschnittstelle

2.3.1 Allgemeiner Aufbau und erforderliche Sichten

Das Layout der Benutzungsschnittstelle soll dem der nachfolgenden Abbildung entsprechen.

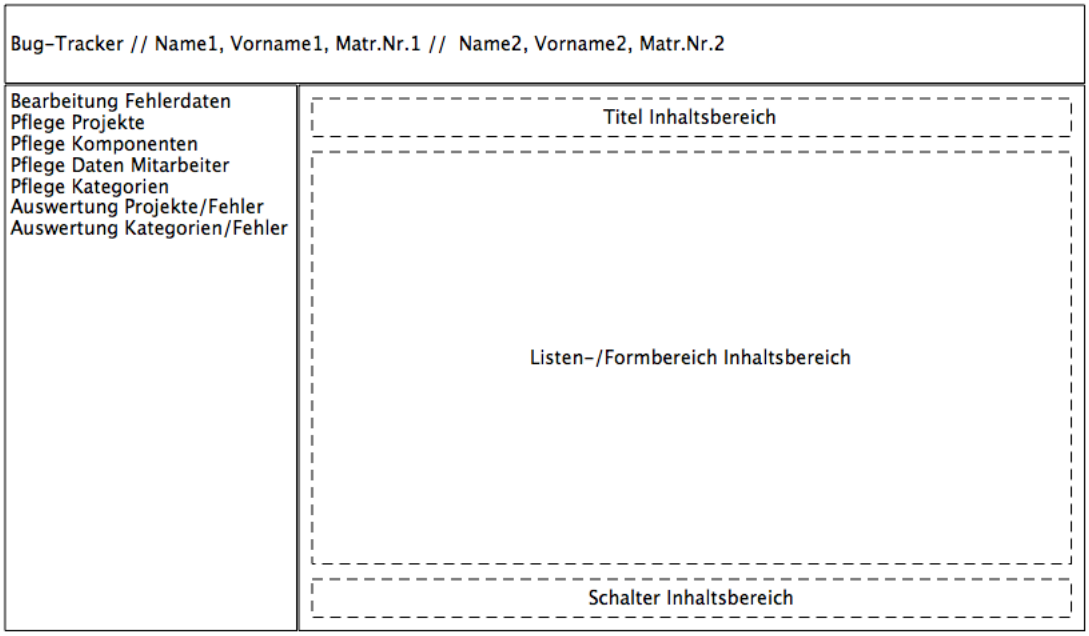

Abbildung 3: Aufbau Benutzungsschnittstelle

Es werden drei Bereiche unterschieden:

- Titelbereich der Anwendung: mit Angabe der Team-Mitglieder (Name, Matrikelnummer)
- Navigationsbereich (links)
- Inhaltsbereich (rechts) mit :
  - Bereich zur Anzeige des jeweiligen Titels
  - Bereich zur Darstellung der Listen und Formulare
  - Bereich zur Darstellung der jeweils erforderlichen Schalter.

Im Navigationsbereich werden folgende Einträge als Links vorgesehen:

- Bearbeitung Fehlerdaten:
  - Listensicht Fehler mit der Möglichkeit der weiteren Filterung nach Fehler (erkannt) / Fehler (behoben)
  - Detailsicht Fehler
    - neue Fehler erhalten immer den Typ/Status "erkannt"
- Pflege Projekte
  - Listensicht Projekte
  - Detailsicht Projekt

- Pflege Komponenten:
  - Listensicht Komponenten zu einem auswählbaren Projekt
  - Listensicht Komponenten (alle)
  - Detailsicht Komponenten
- Pflege Daten Mitarbeiter
  - Listensicht Mitarbeiter mit der Möglichkeit der weiteren Filterung nach QS-Mitarbeiter / SW-Entwickler
  - Detailsicht (QS-Mitarbeiter oder SW-Entwickler)
- Pflege Kategorien:
  - Listensicht Kategorien mit der Möglichkeit der weiteren Filterung nach Fehler / Fehlerursachen
  - Detailsicht (Kategorie Fehler oder Kategorie Fehlerursache)
- Auswertung Projekte/Fehler: im Inhaltsbereich wird die Auswertung als Liste dargestellt
- Auswertung Kategorien/Fehler: im Inhaltsbereich wird die Auswertung als Liste dargestellt.

In den Fällen, in denen die Listensichten gefilter werden können, muss durch einen Texthinweis (z.B. zusätzlich im Titelbereich) auf die Art der Filterung hingewiesen werden.

2.3.2 Strukturierung Datenpflege und Bedienung

Die Benutzungsschnittstelle verwendet bei der Pflege von Daten im wesentlichen zwei verschiedene Darstellungsformen:

- Listensichten:
  - Darstellung der Informationen in einer Liste, die in der Regel als Tabelle (HTML-Element `table`) realisiert wird
  - einzelne Zeilen werden mit einem Mausklick auf die Zeile ausgewählt; die Auswahl mit Radiobuttons oder Checkboxen oder ähnlichen Hilfsmitteln ist **nicht** zulässig
  - Operationen, die auf eine ausgewählte Zeile angewendet werden können (z.B. ändern), werden als separate Schalter angeboten; die Verwendung von Schaltern / Links in jeder Zeile ist **nicht** zulässig
- Detailsichten:
  - Darstellung von Informationen in Formularen
  - Überprüfung von Eingaben mit den in HTML5 vorgesehenen Mitteln sowie zusätzlichen, in javascript realisierten Prüfungen, falls erforderlich.

Der Zusammenhang zwischen Listen- und Detailsichten ist in der folgenden Abbildung wiedergegeben:

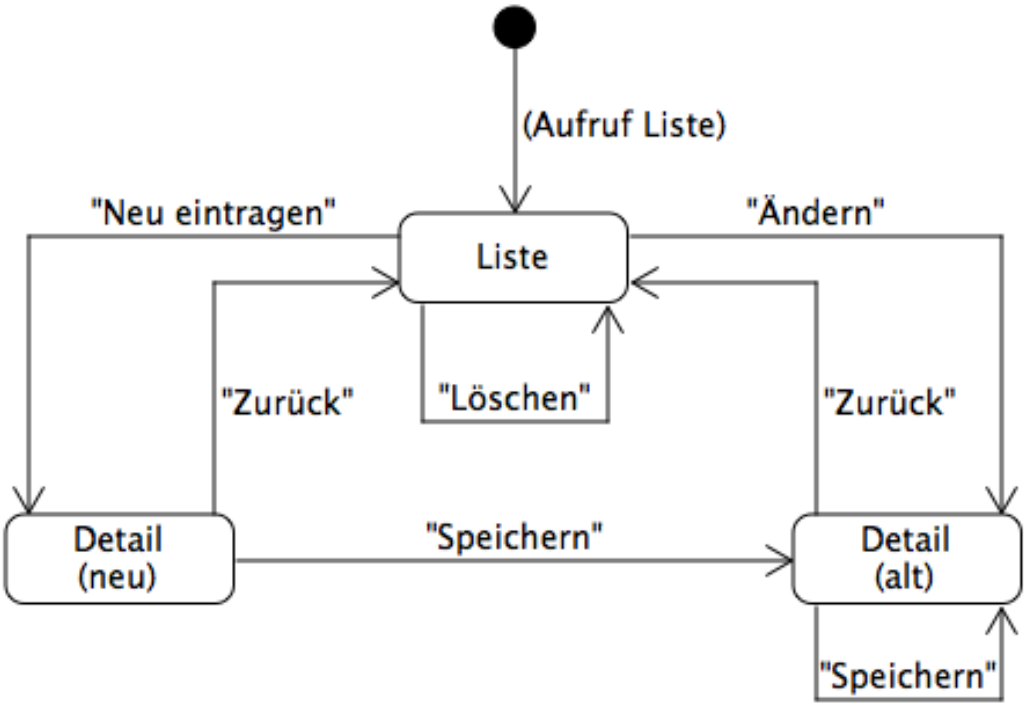

Abbildung 4: Zustandsmodell Listen-/Detailsichten

Erläuterung der Zustandsübergänge:

- im Zustand "Liste":
  - Ereignis "Neu eintragen" (Schalterbedienung) fordert vom Server die Vorgabewerte für neue Daten an und erzeugt ein Formular mit diesen Daten
  - Ereignis "Ändern" (Auswahl Listenelement und Schalterbedienung) fordert vom Server die Daten für den ausgewählten Eintrag an und erzeugt ein Formular mit diesen Daten
  - Ereignis "Löschen" (Auswahl Listenelement und Schalterbedienung) fordert vom Server das Löschen an, als Ergebnis wird die Liste neu aufgebaut oder es wird ein Statuscode des Servers ausgewertet
- im Zustand "Detail (neu)":
  - Ereignis "Zurück" (Schalterbedienung) fordert vom Server die Liste neu an; *vorher* muss geprüft werden, ob es im Formular nicht gespeicherte Änderungen gibt, dem Benutzer muss die Möglichkeit zum Abbrechen des Vorgangs gegeben werden
  - Ereignis "Speichern" (Schalterbedienung): die Daten werden zum Server übertragen und dort in der Datenbasis mit einer neuen Identifikation eingetragen; die neue Identifikation wird als Ergebnis geliefert
- im Zustand "Detail (alt)":
  - Ereignis "Zurück" (Schalterbedienung) fordert vom Server die Liste neu an; *vorher* muss geprüft werden, ob es im Formular nicht gespeicherte Änderungen gibt, dem Benutzer muss die Möglichkeit zum Abbrechen des Vorgangs gegeben werden
  - Ereignis "Speichern" (Schalterbedienung): die Daten werden zum Server übertragen und dort in der Datenbasis mit der vorhandenen Identifikation eingetragen; die bestehende Identifikation wird als Ergebnis geliefert.

Löschoperationen müssen durch den Benutzer bestätigt werden.

2.3.3 Anforderungen an die Gestaltung

Nehmen Sie die Gestaltung mit Hilfe von CSS vor. Berücksichtigen Sie dabei:

- verwenden Sie Farbschemata (Vorder-/Textfarben und Hintergrundfarben), die die Lesbarkeit sicherstellen

- ordnen Sie Schalter in Listen- und Detailsichten so an, dass sich eine übersichtliche und dem Bedienablauf entsprechende Anordnung ergibt
- ordnen Sie Führungstexte (HTML-Element `label`) und die zugehörigen Interaktionselementen übersichtlich an.

Tabellen (HTML-Element `table`) dürfen grundsätzlich **nicht** zur Erzielung eines Layouts verwendet werden. Benutzen Sie dazu andere Elemente, die Sie mit absoluter und/oder relativer Positionierung und/oder mit Flexboxes anordnen. Die Positionierung mit `float` darf **nicht** verwendet werden.

2.4 Weitere Anforderungen

- Webclient:
  - Verwendung HTML5 (XML-konforme Notation)
  - Überprüfung des Markup mit Hilfe der w3c-Validator-Dienste
  - Präsentation mit CSS, ausgelagert in eine externe CSS-Datei
    - die Verwendung von CSS-Frameworks wie z.B. "bootstrap" ist **nicht** zulässig
  - Verwendung javascript / javascript-Bibliothek "jquery"
    - die Verwendung anderer JS-Frameworks ist **nicht** zulässig
    - verwenden Sie Klassen, die Sie ab ES5 mit dem Schlüsselwort `class` definieren können
  - Verwendung der Template-Engine "te.js/tm.js" (**beachten Sie die Erweiterung bei tm.js**)
  - Verwendung des Eventservice "es.js"
- Webserver:
  - Verwendung Python 3
  - Verwendung Framework "cherrypy"
  - eine sinnvolle Aufteilung in Module, die den unterschiedlichen Aufgaben (RequestHandler ["Controller"], Views, Database) gerecht wird; sehen Sie für jede Klasse ein eigenes Modul vor
  - Datenspeicherung mit Hilfe des Python-Modul "json" (siehe Python-Dokumentation) in einzelnen Dateien des serverseitigen Dateisystems in einem Unterverzeichnis "data"

3 Anforderungen an die Dokumentation

Erstellen Sie eine Dokumentation, die Ihre Lösung beschreibt. Verwenden Sie folgende Gliederung:

```
1. Einleitung
   - allgemeine Beschreibung Ihrer Lösung
2. Implementierung des Servers
   2.1 REST-Interface
       - beschreiben Sie alle Anforderungen an den Server, die dem Architekturprinzip REST entsprechen
   2.2 Module
       - geben Sie alle Python-Module an
           - beschreiben Sie die Aufgabe des Moduls
       - geben Sie für jedes Python-Modul an, welche Klassen es enthält
           - beschreiben Sie die Aufgabe der jeweiligen Klasse
       - geben Sie für jede Klasse die öffentlichen Methoden an
           - beschreiben Sie die Aufgabe der jeweiligen Methode
       - beschreiben Sie das Zusammenwirken der Module
   2.3 Datenhaltung
       - beschreiben Sie die Datenstrukturen und die Speicherung in Dateien
3. Implementierung des Clients
   3.1 Klassen
       - geben Sie alle Klassen an
       - beschreiben Sie deren Aufgaben
       - beschreiben Sie das Zusammenwirken der Instanzen der Klassen
```

```
   3.2 Eventservice
       - beschreiben Sie den Einsatz des Eventservice
   3.3 Templateverarbeitung
       - beschreiben Sie die verwendeten Templates
4. Prüfung Markup und Stilregeln
   - beschreiben Sie die Überprüfung des generierten Markups und der CSS-Stilregeln mit den W3C-Validatordiensten.
```

Geben Sie einleitend Ihre Gruppenzugehörigkeit, den Aufbau Ihres Team und das Gültigkeitsdatum der Dokumentation an.

Die Dokumentation wird als utf-8 kodierter Text mit der einfachen Auszeichnungssprache "markdown" erstellt. Mit Hilfe des Werkzeugs "pandoc" kann eine Umsetzung in eine HTML-Datei erfolgen:

```
pandoc -f markdown -t html5 -s <IhreDatei> -o <IhreHTML5Datei>
```

Die in "pandoc" verfügbaren Erweiterungen der Auszeichnungssprache "markdown" müssen genutzt werden.

4 Bewertung / Testat

Zur Bewertung Ihrer Lösung im Hinblick auf die mögliche Erteilung des Testats müssen Sie vorlegen und erläutern:

- den von Ihnen erstellten Quellcode Ihrer Web-Anwendung
- die von Ihnen erstellte Dokumentation einschließlich der Validierung des generierten Markups und der verwendeten CSS-Stilregeln.

Sie müssen die Lauffähigkeit Ihrer Lösungen und die Durchführung der Validierungen nachweisen.

5 Anhang

5.1 Anwendungsbeispiel "lit-x"

Sie erhalten mit dieser Aufgabenstellung das Anwendungsbeispiel "lit-x". Bitte beachten Sie:

- Sie dürfen das Anwendungsbeispiel als Ausgangspunkt Ihrer Implementierung verwenden
- das Anwendungsbeispiel ist *nicht* ausführlich getestet
  - grundsätzlich sollte das Anwendungsbeispiel lauffähig sein
  - Fehlfunktionen müssen Sie ggf. im Rahmen Ihrer Erweiterungen / Änderungen beseitigen
- im Anwendungsbeispiel finden Sie auch die **aktuellen Quellen zu es.js, te.js und tm.js**.

5.2 Verzeichnisstruktur

Verwenden Sie folgende Verzeichnisstruktur und erstellen Sie die angegebenen Dateien:

```
web
   /p3
       /bt                  <--- server.py
           /app             <--- __init__.py, application.py, database.py, view.py, templates.py
           /static          <--- html-Datei(en), css-Datei(en), js-Datei(en); ggf. weitere Unterverzeichnisse
           /data            <--- Daten in JSON-formatierten Dateien; ggf. weitere Unterverzeichnisse
           /doc             <--- Dateien der Dokumentation
           /templates       <--- Vorlagen für te/tm
           /test            <--- curl-Test-Skripte und Testergebnisse
```

5.3 Test REST-Interface mit cURL

Testen Sie das REST-Interface des Webservers mit dem Werkzeug **cURL**. Varianten dieses Werkzeugs für gängige Betriebssysteme und Hinweise zur Anwendung finden Sie unter

- https://curl.haxx.se, insbesondere https://curl.haxx.se/docs/manual.html
- http://alvinalexander.com/web/using-curl-scripts-to-test-restful-web-services
- oder werten Sie eine Suche bei Google z.B. mit den Stichworten "cURL REST" aus.

Erstellen Sie für die einzelnen Ressourcen und Methoden Testskripte an und speichern Sie diese zusammen mit den Ergebnisse im Verzeichnis `test`.

Tipp: Üben Sie den Einsatz von "cURL" zunächst mit dem Anwendungsbeispiel "lit-x".

5.4 Eventservice (es.js)

Das Script wurde von mir entwickelt. Sie dürfen es im Rahmen des WEB-Praktikums verwenden. Eine weitergehende Verwendung ist mir mit abzustimmen.

5.4.1 Publish-Subscriber-Muster

Die Komponente Eventservice implementiert das Publish-Subscriber-Muster:

- `Subscriber` melden sich zum Empfang von Nachrichten an (`subscribe`-Schnittstelle)
- ein `Publisher` veröffentlicht über die `publish`-Schnittstelle des `EventService` eine Nachricht
- die `Subscriber` erhalten dann eine Benachrichtung über ihre `notify`-Schnittstelle
- `Subscriber` beenden den Empfang durch Aufruf der `unSubscribe`-Schnittstelle.

Mit Hilfe dieses Musters ist es möglich, dass Nachrichten von einem Objekt ohne Kenntnis der tatsächlichen Empfänger versendet werden können.

Im folgenden Sequenzdiagramm sind diese Zusammenhänge dargestellt:

- die Verarbeitung der Veröffentlichung (`publish`-Schnittstelle) erfolgt asynchron, daher ist hier auch eine asynchrone Botschaft zwischen `Publisher` und `EventService` eingetragen
- beispielhaft melden sich 2 `Subscriber` zum Nachrichtenempfang an
- eine Nachricht wird durch den Aufruf der `notify`-Schnittstellen an die `Subscriber` weitergeleitet
- die `Subscriber` beenden den Nachrichtenempfang.

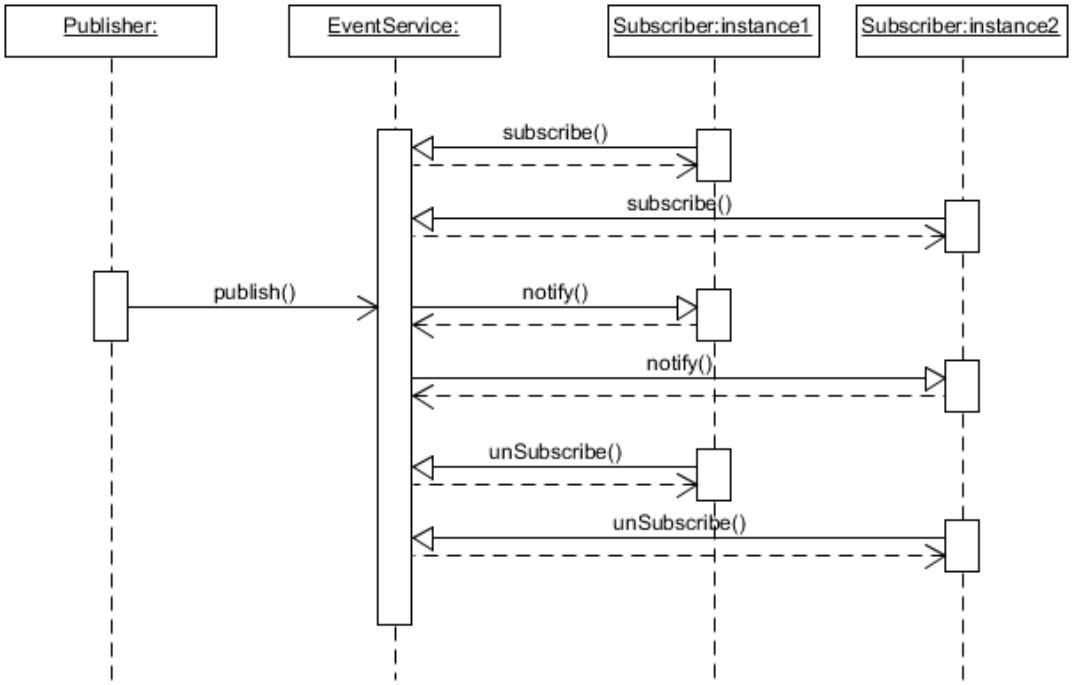

Abbildung 5: Sequenzdiagramm Publish-Subscriber

Zur Notation:

- durchgezogene Linie mit offener Pfeilspitze: asynchroner Aufruf einer Methode = Nachricht asynchron senden
  - es wird dann auch keine Antwort erwartet
- durchgezogene Linie mit geschlossener Pfeilspitze: synchroner Aufruf einer Methode = Nachricht synchron senden
  - die Antwort wird als gestrichelte Linie mit offener Pfeilspitze dargestellt.

5.4.2 Implementierung

Der `Eventservice` wird in der Klasse `EventService_cl` implementiert. Diese enthält die Methoden

- `constructor`
- `subscribe_px`
- `unSubcribe_px`
- `publish_px`.

Beim Registrieren (`subscribe_px`) und Aufheben der Registrierung (`unSubscribe_px`) muss ein `Subscriber` den Verweis auf sich selber sowie die Bezeichnung der Nachricht übergeben. Er erhält dann beim Auftreten einer passenden Nachricht über seine Schnittstelle `notify_px` einen Verweis auf sich, die Bezeichnung der Nachricht sowie Daten zur Nachricht.

Die Funktionen `each`, `findAll`, `compact` dienen der Vereinfachung der Implementierung.

Asynchrone Aufrufe der `notify_px`-Schnittstelle werden durch das `hashchange`-Ereignis ermöglicht: bei jeder Änderung des "Fragments" des aktuell vorliegenden URI (auch als "Hash-Wert" bezeichnet, daher der Ereignisname) wird die entsprechende Ereignisbehandlungsfunktion aufgerufen - hier die Methode `send_p`.

5.4.3 Beispiel Verwendung Eventservice

```
'use strict'

let APP = {};
APP.es_o = new EventService_cl();

class myClass_cl {
   constructor () {
      APP.es_o.subscribe_px(this, 'app');
   }
   notify_px (self_opl, message_spl, data_apl) {
      switch (message_spl) {
      case 'app':
         switch (data_apl[0]) {
         case 'work.done':
            console.log(data_apl[1]); // TODO: müsste man vorher auf Existenz prüfen!
            break;
         default:
            console.warning('[Application_cl] unbekannte app-Notification: '+data_apl[0]);
            break;
         }
         break;
      default:
         console.warning('[Application_cl] unbekannte Notification: '+message_spl);
         break;
      }
   }
}

let myObject_o = new myClass_cl();

APP.es_o.publish_px('app', ['work.done', 'success']);
```