Dokumentation des PoCs zur Datenmodellierung

Die gewünschte Datenmodellierung waren ineinander verschachtelte Datensätze, die in der Datenbank so abgespeichert und ausgegeben werden müssen, dass dabei eine Baumstruktur entstehen kann. Somit hat ein Datensatz ein Elternelement und mehrere Kinderelemente.

Als Datenbanksystem wurde sich für MongoDB entschieden, da dafür schon Erfahrung besteht. Zur Zugriffsschnittstelle wurde mongoose gewählt, durch die hohe Community und umfangreichen Dokumentation.

Die Definition des Schemas eines Datensatzes in ausgedünnter Form sieht wiefolgt aus:

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var ObjectId = Schema.ObjectId;

var beitragSchema = Schema({
  _parent: { type: ObjectId, ref: 'Beitrag' },
  title: String,
  children: [{ type: ObjectId, ref: 'Beitrag' }]
});

var Beitrag = mongoose.model('Beitrag', beitragSchema);
```

Das Abspeichern der Datensätze hat sich nicht als schwer erwiesen, da zu Referenzierungen in mongoose Erfahrung vorhanden war. Gemerkt wurde, dass hierbei darauf geachtet werden muss, dass sowohl der Datensatz mit korrekter ElternID abgespeichert wird, als auch die ID des Eintrags in die Liste der Kinder des Elternteils eingefügt werden muss.

```
var eltern = mongoose.Object(); // es wird angenommen, dass dieses
Objekt vorhanden ist

var beitrag = new Beitrag({
  _parent: eltern._id,
  title: "Ein Beitrag"
});

beitrag.save(function () {
  eltern.children.push(beitrag);
  eltern.save();
});
```

Ein Objekt ohne Elternteil (folglich ein Wurzelobjekt) wird als _parent *NULL* haben.

Anhand diesen Beispielcodes wurden mehrere Beiträge aufeinander referenziert abgespeichert. Für die Ausgabe müssen jene Referenzen *populated* werden, d.h. die Referenz mit dem Datensatz ersetzt werden.

In MongoDB/mongoose sind tiefe *populations* nicht integriert, folglich müssen bis zu einer bestimmten Ebene die *population* festgelegt werden.

```
Beitrag.find({ _parent: null }) // nur Wurzelobjekte finden
  .lean()
  .populate({ path: 'children' }) // erste Ebene populaten
  .exec(function(err, result) {

    // zweite Ebene populaten
    Beitrag.populate(result, { path: 'children.children' },
     function(err, _result) {

      // dritte Ebene populaten
      Beitrag.populate(result, { path:
'children.children.children'
       }, function(err, _result) {

        // "result" ist hier bis zur 3. Ebene populated

      });
    });
  });
```

Das Ausarbeitung dieser Funktionalität war schwieriger als angenommen, da eine saubere Methodik gesucht wurde. Verschiedene Node-Module als Erweiterung von mongoose, die diese Funktionalität anbieten, wurden in Betracht gezogen. Jene Node-Module verwenden die gleiche Funktionalität und konnten somit verworfen werden.

Der `lean()`-Modus ist hierfür notwendig, was jedoch einen Vorteil mit sich bringt, da durch ihn die Rechenlaufzeit ebenfalls verkürzt wird. Möchte man weiter als drei Ebenen *populaten*, so können auf diese Weise weitere *populates* ineinander verschachtelt werden.

Durch die Ausgabe von result konnte festgestellt werden, dass die Ausgabe wie gewünscht erfolgen konnte.

```
➜ poc-api  node index.js
[ { _id: 563ce688c1b550192c612e94,
    title: 'Wirtschaftsinformatik',
    children:
     [ { _id: 563ce688c1b550192c612e97,
         _parent: 563ce688c1b550192c612e94,
         title: 'Beitrag in WI Thread',
         children:
          [ { children: [],
              __v: 0,
              title: 'Hier, ein Kommentar',
              _parent: 563ce688c1b550192c612e97,
              _id: 563ce688c1b550192c612e9a } ],
         __v: 1 } ] },
  { _id: 563ce688c1b550192c612e93,
    title: 'Medieninformatik',
    children:
     [ { _id: 563ce688c1b550192c612e95,
         _parent: 563ce688c1b550192c612e93,
         title: 'Ein Beitrag',
         children:
          [ { children: [],
              __v: 0,
              title: 'Ein Kommentar',
              _parent: 563ce688c1b550192c612e95,
              _id: 563ce688c1b550192c612e98 },
            { children: [],
              __v: 0,
              title: 'Weiterer Kommentar',
              _parent: 563ce688c1b550192c612e95,
              _id: 563ce688c1b550192c612e99 } ],
         __v: 2 },
       { _id: 563ce688c1b550192c612e96,
         _parent: 563ce688c1b550192c612e93,
         title: 'Weiterer Beitrag',
         children: [],
         __v: 0 } ] } ]
```

Referenziert werden tatsächlich nur die IDs.

▶ **_id**	ObjectId("563ce688c1b550192c612e93")	Object id
▶ **_id**	ObjectId("563ce688c1b550192c612e94")	Object id
▼ **_id**	ObjectId("563ce688c1b550192c612e95")	Object id
_id	ObjectId("563ce688c1b550192c612e95")	Object id
_parent	ObjectId("563ce688c1b550192c612e93")	Object id
title	Ein Beitrag	String
▼ **children**		Array, 2 items
0	ObjectId("563ce688c1b550192c612e98")	Object id
1	ObjectId("563ce688c1b550192c612e99")	Object id
__v	2	Integer
▶ **_id**	ObjectId("563ce688c1b550192c612e96")	Object id
▶ **_id**	ObjectId("563ce688c1b550192c612e97")	Object id
▶ **_id**	ObjectId("563ce688c1b550192c612e98")	Object id
▶ **_id**	ObjectId("563ce688c1b550192c612e99")	Object id
▶ **_id**	ObjectId("563ce688c1b550192c612e9a")	Object id