# Software Architektur Dokument
# Projekt BierIdee

Danilo Bargen, Christian Fässler, Jonas Furrer

2. April 2012

# Inhaltsverzeichnis

# Änderungshistorie

| Version | Datum | Änderung | Person |
|---|---|---|---|
| v1.0 | 02.04.2012 | Dokument erstellt | dbargen |

# 1 Einführung

## 1.1 Zweck

Dieses Dokument beschreibt die Softwarearchitektur des Projektes BierIdee.

## 1.2 Gültigkeitsbereich

Die Gültigkeit des Dokumentes beschränkt sich auf die Dauer des SE2-Projekte Modules FS2012.

## 1.3 Referenzen

- REST.Interface.pdf

## 1.4 Übersicht

TODO

# 2 Systemübersicht

Unsere Systemarchitektur gliedert sich in drei Teilbereiche: Die Datenbank, die REST API und die Clientanwendung.

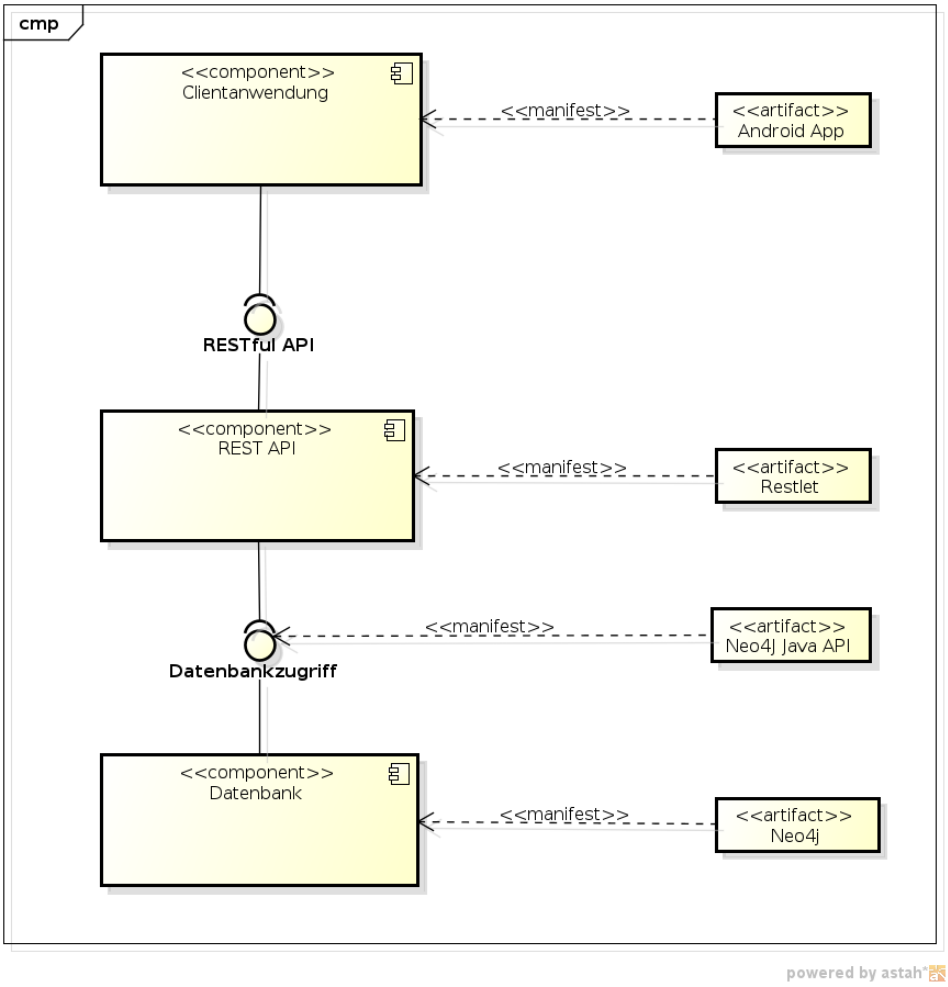


Abbildung 1: Komponentendiagramm

## 2.1 Komponenten

### 2.1.1 Datenbank

Als Datenbank kommt die Graphendatenbank Neo4j[1] zum Einsatz. Eine Graphendatenbank hat den Hauptvorteil, dass man sehr gut mit den Relationen arbeiten kann und so beispielsweise die Bier-Empfehlungen durch geschickte Traversierung realisieren kann.

[1] `http://neo4j.org/`

### 2.1.2 REST API

Für das Zwischenglied zwischen Daten und Client kommt ein RESTful[2] Web Service zum Zuge. Diese REST API wird mithilfe von Restlet[3] realisiert. Die Domainobjekte werden in Form von abstrakten Ressourcen mit entsprechenden Repräsentationen zur Verfügung gestellt.

Die REST-Ressourcen werden im Dokument *REST.Interface.pdf* näher spezifiziert.

### 2.1.3 Clientanwendung

Die Android Clientanwendung greift auf die REST API zu und stellt die Informationen auf sinnvolle Art und Weise dar.

## 2.2 Schnittstellen

### 2.2.1 Datenbankzugriff

Der Datenbankzugriff geschieht über die eingebaute Java-Schnittstelle von Neo4j. Dadurch wird die Verwendung von JDBC oder eines OR Mappers unnötig. Datenbanknodes können entweder via objektorientierter Syntax oder mithilfe einer domainspezifischen Abfragesprache namens *Cypher* abgefragt werden.

### 2.2.2 REST API / HTTP

Die RESTful API wird von Restlet bereitgestellt. Die Domainobjekte werden in Ressourcen gegliedert und via HTTP im JSON- oder XML-Format ausgegeben.

# 3 Architektonische Ziele & Einschränkungen

TODO Beschreibt die Softwareanforderungen und Objekte, welche einen Einfluss auf die Architektur haben (z.B. Safety, Security, Privacy, Distribution, usw.); Beinhaltet auch eine Beschreibung von Design und Implementationsstrategie, Entwicklungstools, usw

# 4 Logische Architektur

TODO Beschreibung der logischen Struktur des Projekts. Pro Subsystem/Package ein einzelner Abschnitt und ein Übersichtsdiagramm über die einzelnen Subsyste-

[2] `http://en.wikipedia.org/wiki/Representational_state_transfer#RESTful_web_services`
[3] `http://www.restlet.org/`

me/Packages. Aufteilung in Subsysteme/Packages (zum Beispiel: 3-Layer-Architektur mit GUI, Problem Domain und Datenhaltung).

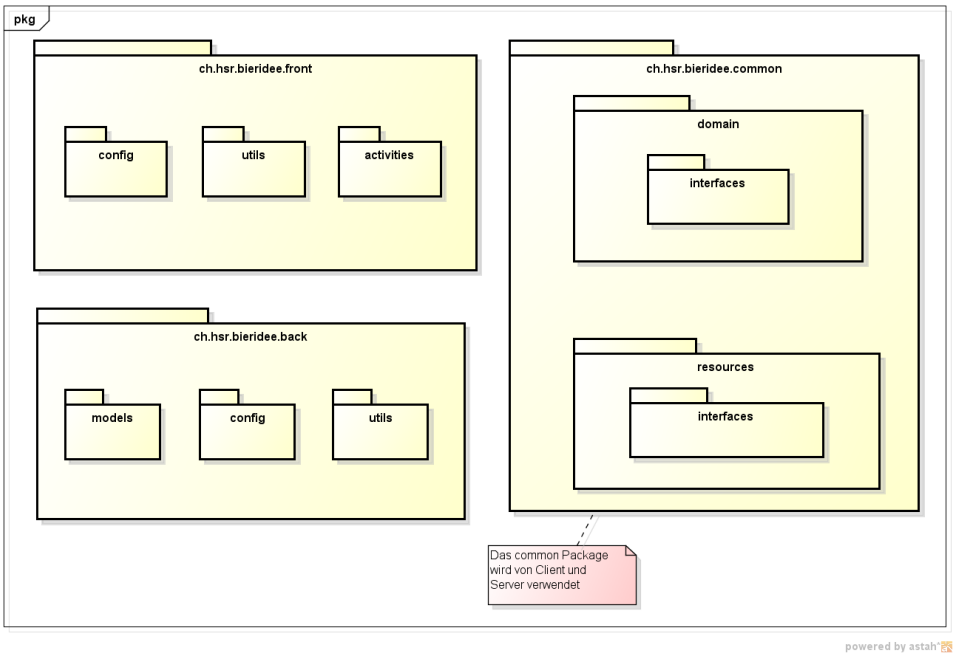


Abbildung 2: Package Diagramm

# 5 Backend Architektur

Die Backend-Architektur ist ein ziemlich wichtiger Teil des Architekturdokumentes. Deshalb nachfolgend ein detailliertes Backend-Architektur-Klassendiagramm am Beispiel eines Bier-Objektes.

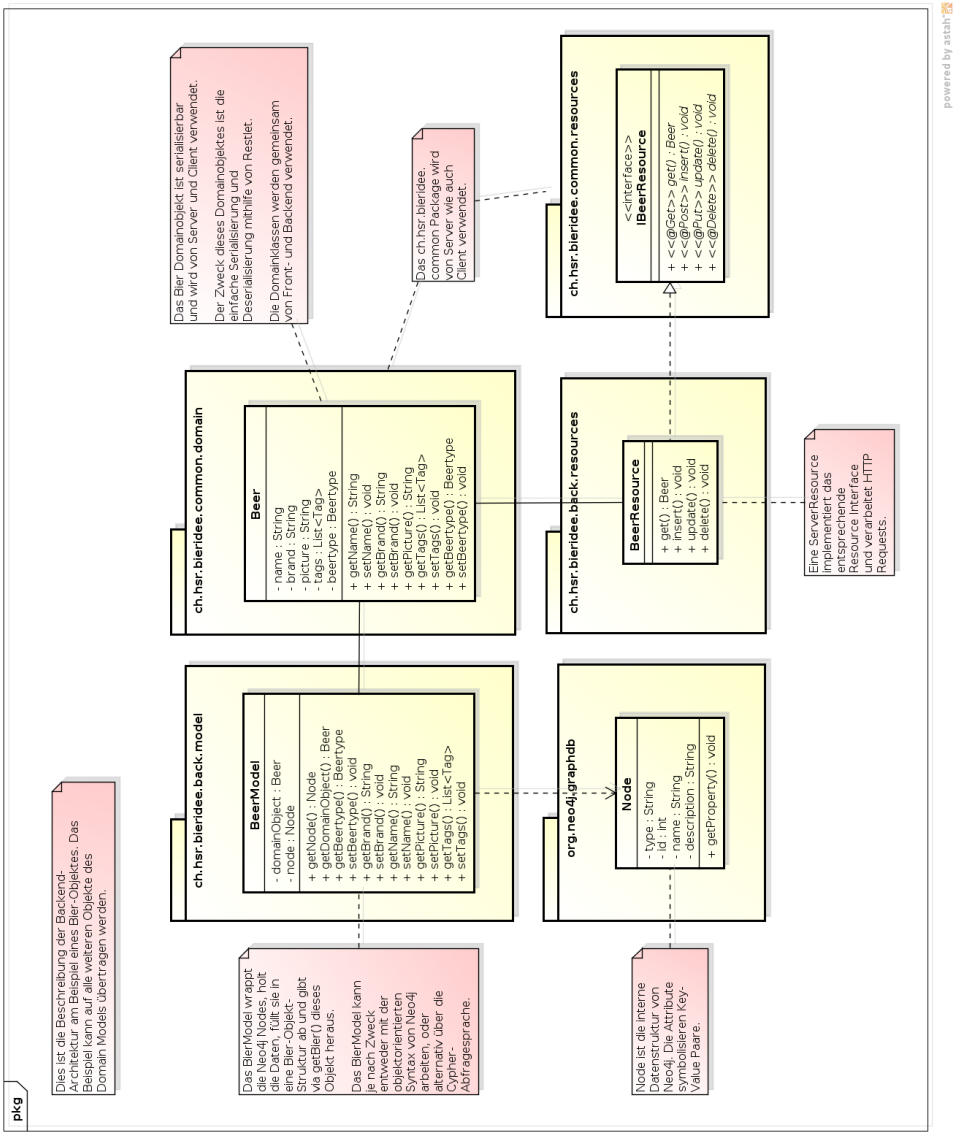


Abbildung 3: Backend Architektur

# 6 Prozesse und Threads

## 6.1 Datenbank

Die Neo4j Datenbank garantiert die ACID Prinzipien. Sämtliche Datenzugriffe erfolgen in Transaktionen, somit ist die Integrität der Datenbank sichergestellt. Neo4j ist komplett thread safe implementiert. Die Datenbank wird im embedded Modus verwendet und kann direkt über eine Java API angesprochen werden.

## 6.2 Server API

Die Server API ist multithreaded. Für jede Clientanfrage wird ein Worker Thread erstellt welcher den Request bearbeitet.

Die benötigten Ressourcen-Handler werden bei jeder Anfrage neu instanziert, somit müssen diese Komponenten nicht thread safe implementiert sein.

## 6.3 Android App

Die UI ist singlethreaded. Langlaufende Operationen werden in asynchrone Background-Threads ausgelagert und benachrichtigen das UI mithilfe von Callbacks.

# 7 Deployment

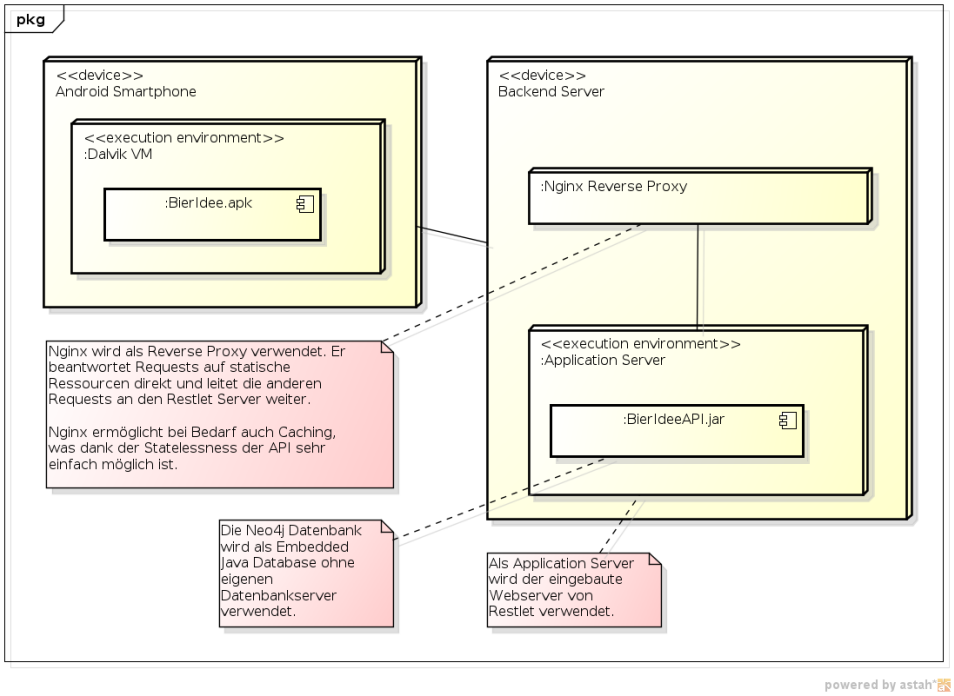


Abbildung 4: Deployment Diagramm

## 7.1 Frontend

Die Frontend-Applikation wird im Rahmen dieses Projektes manuell als `.apk`-Datei auf die Testgeräte kopiert werden. Falls das Projekt weitergezogen wird, wäre eine

Publikation im Android Market gut denkbar, für den Moment wird aber darauf verzichtet.

### 7.2 Backend

Das Backend wird auf einem Debian Linux Server deployed. Die Application Server Instanz läuft mithilfe von Java 1.6 und Restlet ohne zusätzliche Software.

Zur Überwachung des Systemprozesses wird Supervisor verwendet (nicht im Deployment Diagramm erwähnt, da kein integraler Bestandteil). Supervisor überwacht den Systemprozess und startet ihn, falls er abstürzt, sofort neu.

Vor den Application Server wird ein Reverse Proxy - konkret Nginx - gesetzt. Dieser nimmt alle Requests entgegen und entscheidet, ob es sich dabei um statische oder um dynamisch generierte Daten handelt. Statische Daten - beispielsweise Bilder - werden direkt zurückgeliefert, während die anderen Requests an den Application Server weitergegeben werden. So kann auch ein Caching problemlos implementiert werden, was den Application Server entlastet und die Skalierbarkeit stark erhöht.

## 8 Datenspeicherung

Die Daten werden in einer Graphendatenbank (konkret Neo4j) abgelegt. Diese besteht grundsätzlich nur aus zwei Arten von Objekten - Nodes und Relationen. Mithilfe von diesen zwei Objekten kann die gesamte Datenstruktur sehr flexibel abgebildet werden; Abfragen können mit gezielter Traversierung durch eine domainspezifische Abfragesprache namens *Cypher* ausgedrückt werden.

Nachfolgend das allgemeine Datenbankdiagramm sowie ein Diagramm eines Beispielfalles.

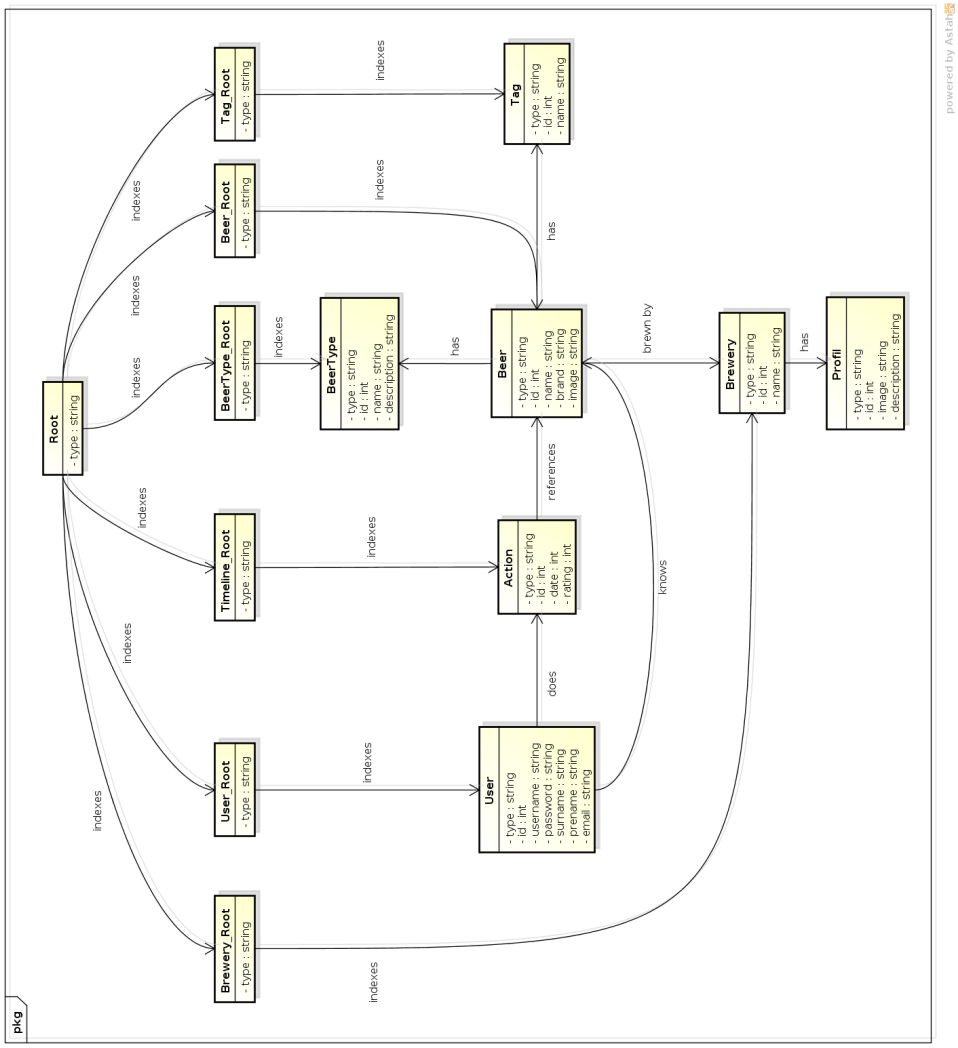


Abbildung 5: Allgemeines Datenbankdesign

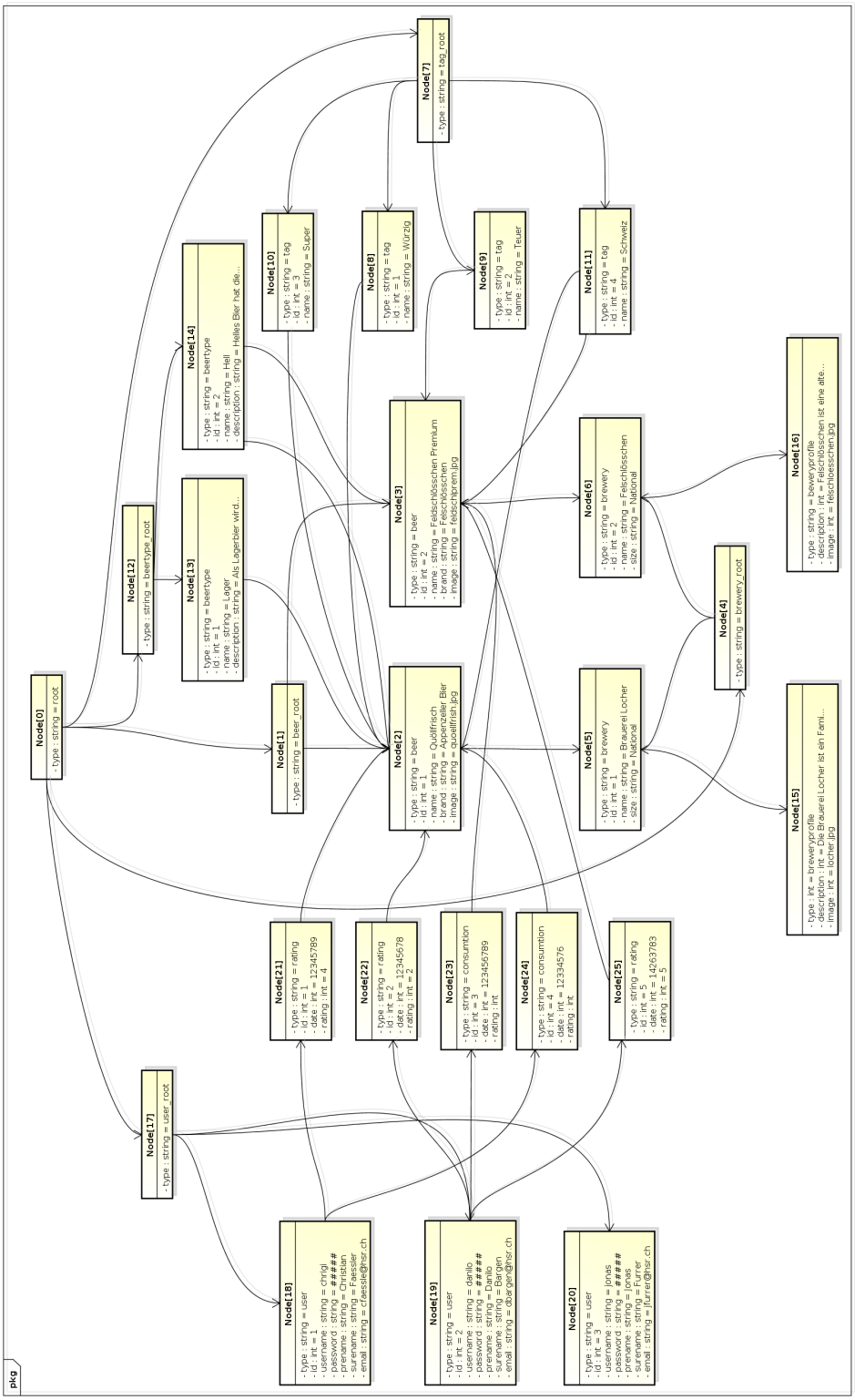


Abbildung 6: Datenbank Beispielfall

# 9 Grössen und Leistung

Im Rahmen dieses Projektes ist das System und die serverseitige Umgebung für den Einsatz mit bis zu 50 Benutzern, rund 100 Bieren und 20 Brauereien ausgelegt. Es sollte aber skalierbar sein, so dass mit besserer Serverumgebung erheblich mehr Benutzer bedient werden könnten.

Die Architektur wird so ausgelegt, dass so wenig Rechenleistung wie möglich auf der Clientseite benötigt wird. Das heisst, sämtliche Logik und Aufbereitung der Daten wird auf der Serverseite implementiert. Da clientseitig die Unterschiede in der verwendeten Hardware sehr gross sein können (Smartphones, Tablets, Netbooks), können keine einheitlichen Anforderungen zur Rechenkapazität auf den Geräten gestellt werden. Desweiteren muss heutzutage bei der Entwicklung von Mobile Apps stark auf den Energieverbrauch geachtet werden. Mit der zentralisierten Ausführung rechenintensiver Aufgaben sind die verfügbaren Ressourcen bekannt und die Software kann entsprechend adäquat entwickelt werden.