Software Architektur Dokument
Projekt BierIdee

Danilo Bargen, Christian Fässler, Jonas Furrer

5. Mai 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
- Evaluation.Datenbank.pdf
- Authentication.pdf

1.4 Übersicht

Das System der Bieridee kann in drei Kategorien/Teilbereiche unterteilt werden: Die Datenbank, die REST API und die Clientanwendung. Zwischen der API und der Android-Clientanwendung wird ausschliesslich via HTTP kommuniziert. Um auf die API zuzugreifen, muss man sich mit einem API-Token authentifizieren. Logisch werden die Datenbank- und API-bezogenen Klassen in einem gemeinsamen Package geführt, das selbe gilt für das Frontend. Zusätzlich werden jedoch alle gemeinsam verwendeten Klassen und Interfaces in einem separatem, gemeinsam zugänglichen Package gehalten. Beim Deployment wird die API direkt über den eingebauten Webserver von Restlet gestartet, ohne Servlet-Container. Es wird aber aus Performance- und Optimierungsgründen ein Reverse Proxy davorgesetzt. Alle Daten werden im Backend persistent in einer Graphendatenbank gespeichert und durch geschickte Traversierung und eingebaute Algorithmen abgefragt. Das System soll in der Testphase bis zu 50 gleichzeitige Nutzer bedienen können.

2 Systemübersicht

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

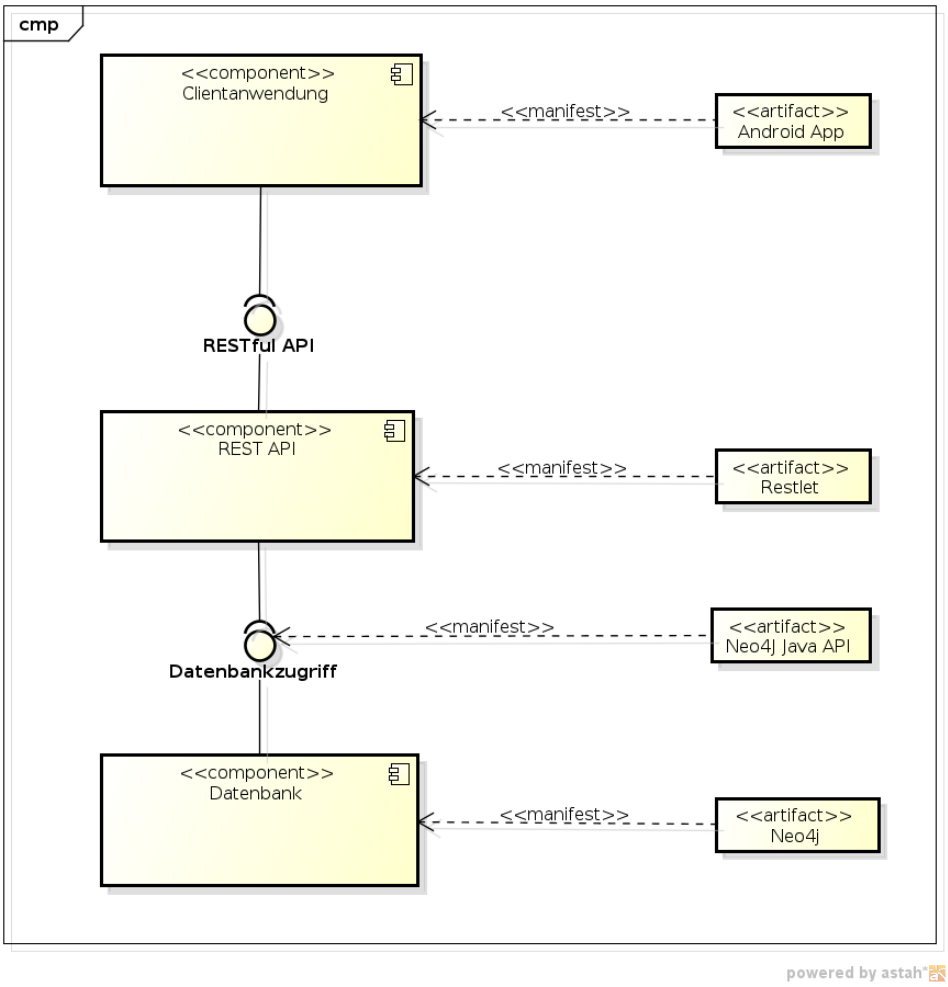

Abbildung 1: Komponentendiagramm

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

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

realisieren kann. Näheres zum Entscheid zur Verwendung einer Graphendatenbank siehe Dokument Evaluation.Datenbank.pdf

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

3.1 Safety / Security

Die API wird nicht öffentlich/frei zugreifbar sein. Das Sicherheitskonzept ist zweistufig aufgebaut. Einerseits erzeugt die API sogenannte API-Tokens, mit welchem sich die App als „trustful“ App identifiziert.

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

In einem zweiten Schritt authentifiziert und autorisiert sich der Benutzer der Android App mittels Benutzernamen und Passwort.

Das genaue Authentifizierungs bzw. Signierungsverfahren ist im Dokument Authentication.pdf dokumentiert.

Security-Steps:

1. App Authentifizierung mittels App-Token
2. User-Authentifizierung mittels Benutzername und Passwort
3. Autorisierung für gewünschte Operationen

Alle Schritte werden bei jeder Anforderung einer Ressource (http Request) durchgeführt, um das Prinzip der *Statelessness* von RESTful APIs zu gewährleisten.

Auf Datenbankebene wird kein Rechtesystem implementiert. Die Datenbank ist auch nicht direkt ansprechbar, daher kann das ganze Berechtigungssystem in der REST API realisiert werden.

4 Logische Architektur

Die grundsätzliche Aufteilung in Subsysteme in diesem Projekt wird bereits im Kapitel *Systemübersicht* beschrieben, deshalb wird hier primär auf die Package-Architektur eingegangen.

Die logische Architektur des Gesamtprojektes wird in drei Kategorien bzw. Packages unterteilt – *front*, *back* und *common*. Alle drei Packages sind Unterpackages von *ch.hsr.bieridee*.

Neben den Frontend- und Backend-Packages wird ein Package *ch.hsr.bieridee.common* erstellt, welches die gemeinsamen Teile des Front- und Backends beinhaltet (konkret die Domainobjekte und -interfaces, sowie die Ressourcenobjekte und -interfaces).

Im Backend wird die Datenhaltung vom Package *ch.hsr.bieridee.back.models* übernommen, welches die entsprechende Datenbanknode kapselt. Desweiteren gibt es ein *config*- und ein *utils* Package.

Ähnliches gilt für das Frontend. Momentan sind die Packages *config*, *utils* und *activities* als Unterpackages von *ch.hsr.bieridee.front* geplant.

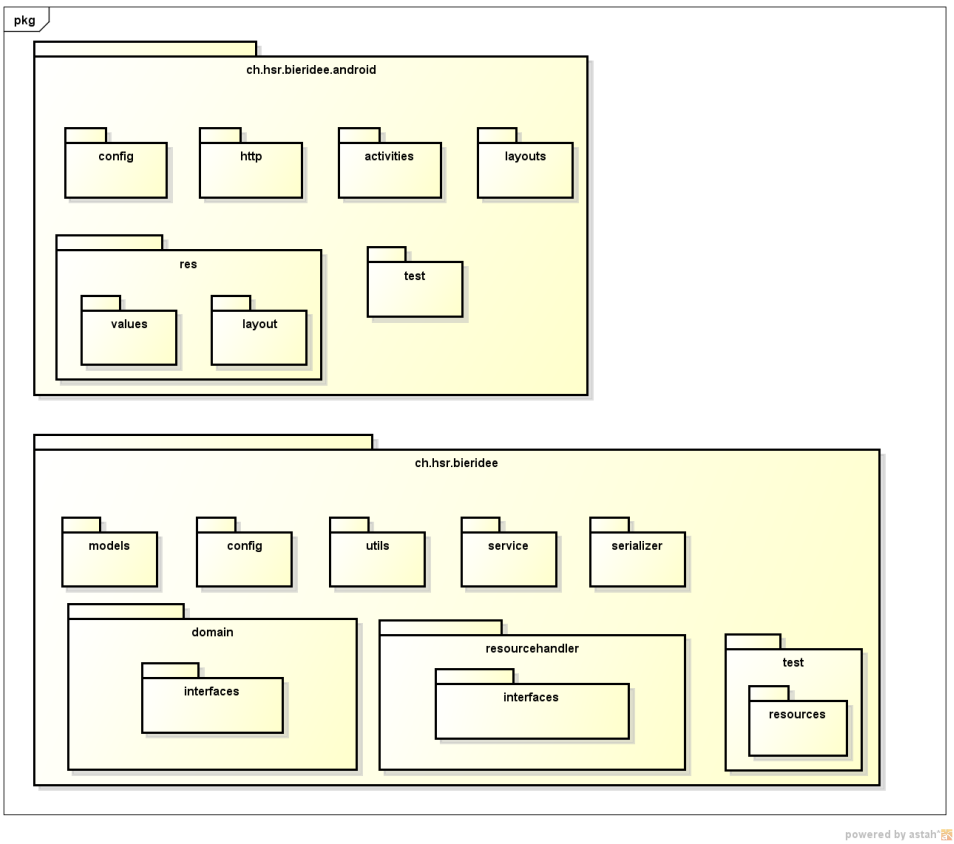

Abbildung 2: Package Diagramm

Selbstverständlich wird die Android-Applikation nach dem Model-View-Controller Modell implementiert. Dies ist jedoch zusammen mit dem Ressourcen-Management (Templates, Strings, Grafiken etc) ein integraler Bestandteil des Android SDKs, deshalb wird es hier nicht genauer erörtert.

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.

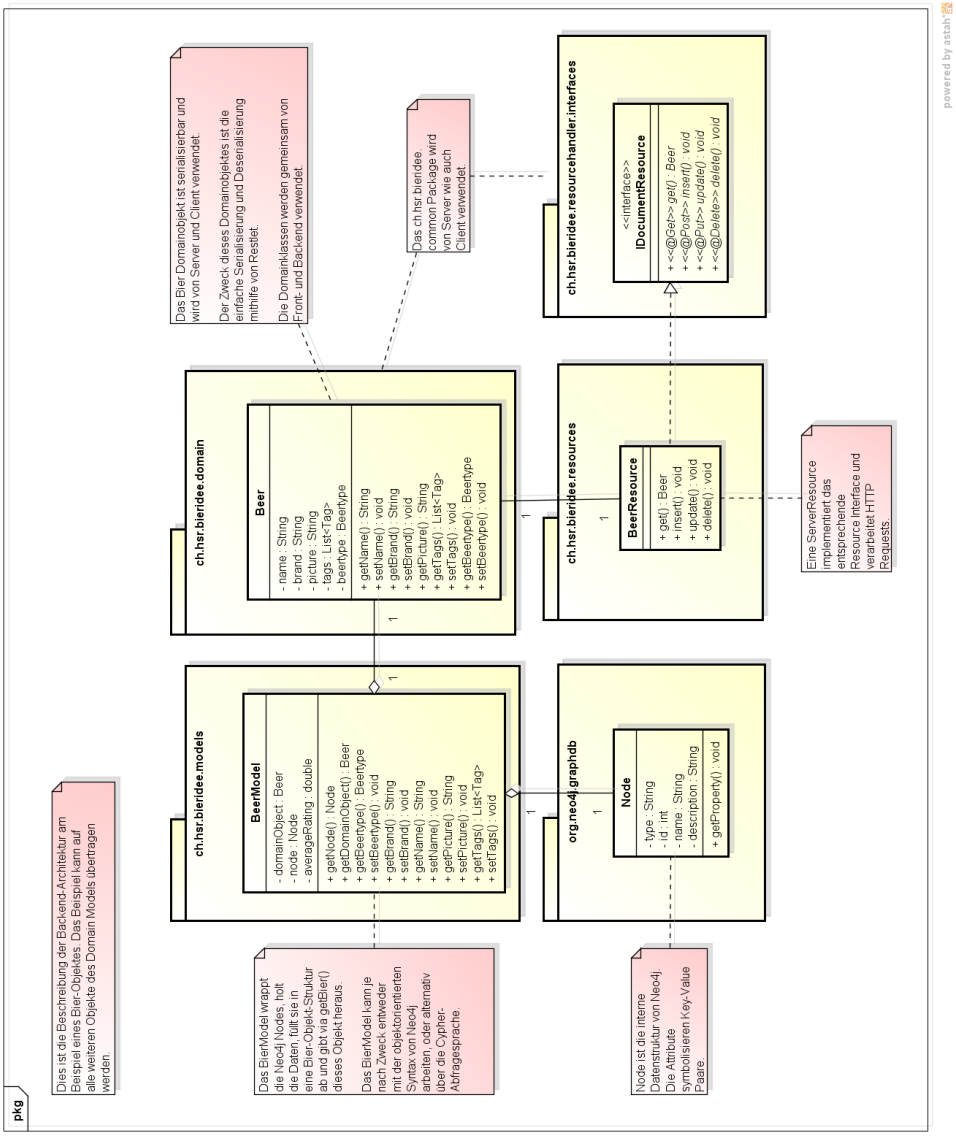

Abbildung 3: Backend Architektur

6 Frontend Architektur

Dieses Diagramm zeigt einen Ausschnitt aus der Frontendarchitektur am Beispiel der Bierliste. Die Architektur ist grösstenteils durch das Android SDK und ihre Best-Practice Richtlinien vorgegeben.

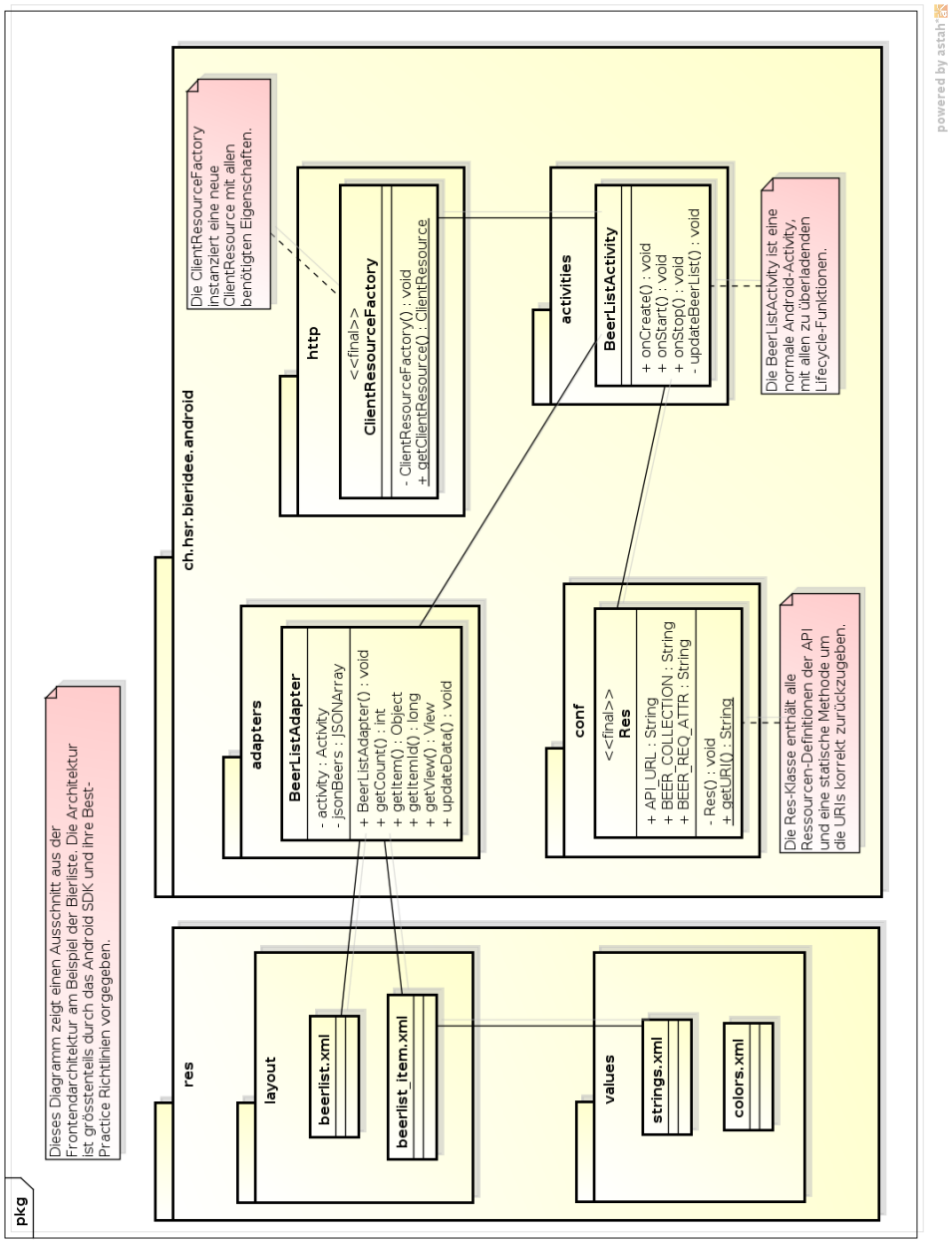

Abbildung 4: Frontend Architektur

7 Prozesse und Threads

7.1 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.

7.2 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.

7.3 Android App

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

8 Deployment

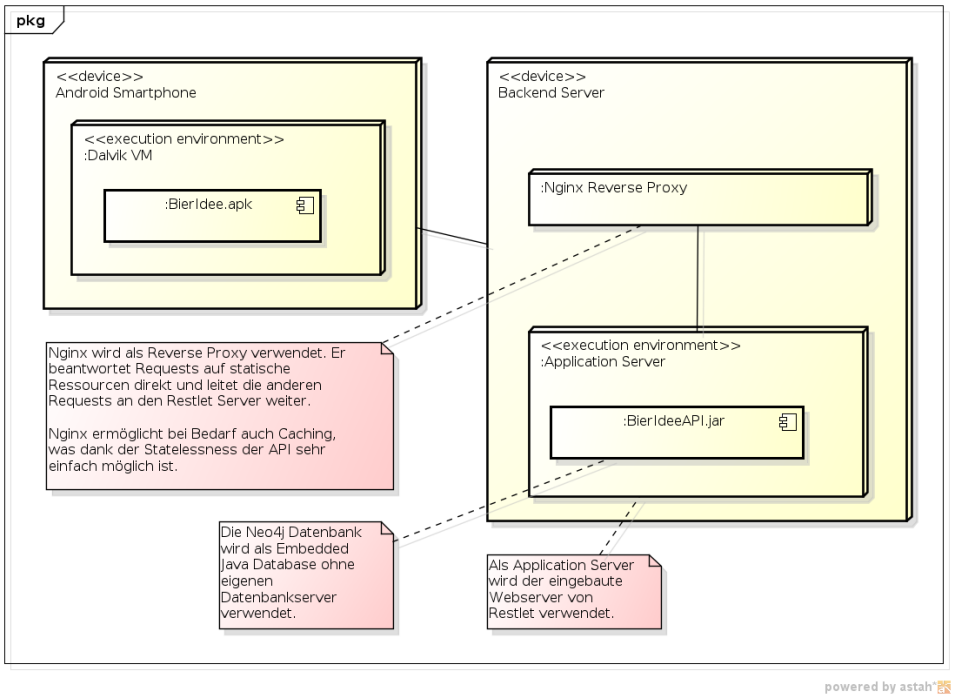

Abbildung 5: Deployment Diagramm

8.1 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.

8.2 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.

9 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.

Der Evaluationsbeschrieb der Datenbank findet sich im Dokument *Evaluation.Neo4j.pdf*.

Nachfolgend das allgemeine Datenbankdiagramm sowie ein Diagramm eines Beispielfalles:

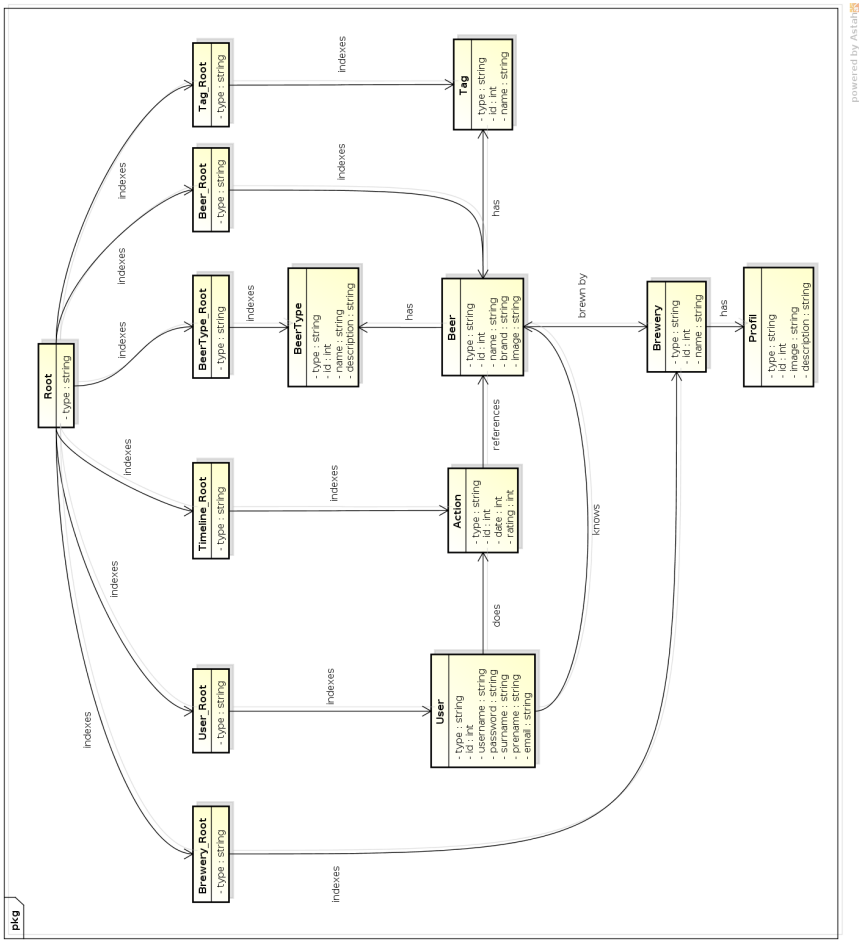

Abbildung 6: Allgemeines Datenbankdesign

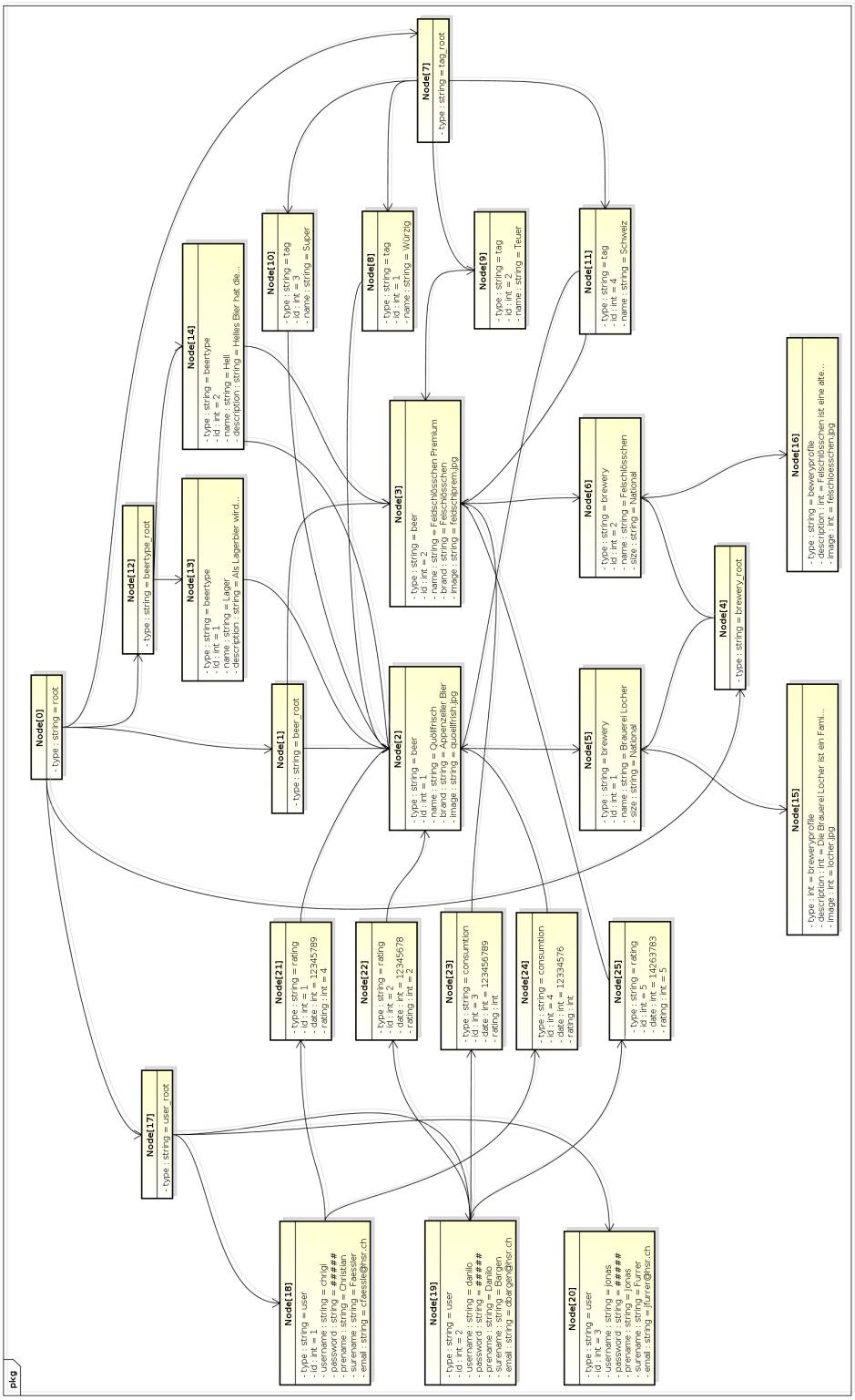

Abbildung 7: Datenbank Beispielfall

10 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.