Fachhochschule Köln - Campus Gummersbach
Web-basierte Anwendungen 2

Logbuch

GeoCatching

Pascal Kottmann
Sommersemester 2013

Inhaltsverzeichnis

15.04.2013 – Ideenfindung

Brainstorming für Projektideen und Abschätzung der Umsetzbarkeit.

Entscheidung für ein Programm oder eine App für „Geocacher“, oder die die es werden wollen, mit dem es möglich ist einen Account und eigene Geocaches zu erstellen und sich bereits erstellte Geocaches in einem Bestimmten Umkreis und unter Bestimmten Bedingungen Anzeigen zu lassen.

Des Weiteren besteht die Möglichkeit ein bestimmtes Gebiet oder auch mehrere Gebiete zu abonnieren, wird dann ein Cache in diesem Gebiet erstellt bekommt der Abonnent eine Mitteilung darüber das ein Cache erstellt wurde und kann sich dann unter Feeds die Details der Abonnierten Caches in der Umgebung Ansehen.

22.04.2013 – Meilenstein: Kommunikationsabläufe und Interaktionen

Mithilfe der XML Dateien die einzeln vom Server abgefragt und verarbeitet werden können wird der Anforderungsteil der Synchronen Kommunikation abgedeckt, dies werden wir Beispielsweise bei der Nutzererstellung und dem Abrufen der Caches unter bestimmten Anforderungen genutzt.

Mithilfe von XMPP wollen wir die Asynchrone Synchronisation verarbeiten um die Nutzer über neue Caches in ihrem Abonnierten Bereich zu informieren, hierbei wird der bei der Abonnement Erstellung eingetragene Ort inkl. Des maximalen Umkreises beachtet, was dem Nutzer erspart immer wieder selbst nachsehen zu müssen ob seit seinem Letzten Besuch neue Caches erstellt wurden.

26.04.2013 - Identifizierung von Ressourcen und Erstellen der XML-Schemata

Wir benötigen für unser Projekt lediglich Zwei XML Dateien, diese wären die Cachelist.xml sowie die User.xml

CacheList – XML

Kommen wir nun zu dem Kernteil unseres Projekts. Den Caches. Hierfür haben wir eine validierbare Cachelist erstellt, welche alle für den Benutzer gewünschten Geocaches enthält. Die Liste enthält einzelne Cache-Elemente zu diesem Cache. Diese Cachelist soll über das Internet an Client gesendet werden. Diese kompakte Informationsstruktur bietet den Vorteil der einfachen Handhabung. Alle relevanten Informationen sind in einer einzigen Datei gespeichert und können so komplett übertragen werden. Die Endgeräte erhalten jeweils nur die Liste, welche sie auch benötigen, was den Datenverkehr auf ein Minimum reduziert. Leider hat die kompakte Struktur den Nachteil, dass das Dokument sehr verschachtelt ist. Wichtige Elemente, die vom Aufbau her immer gleich sind, haben wir als Attribute definiert. Hierzu fällt z.B:

<location lat="*50.98817*" lon="*7.83573*"></location>
Hier sind die Attribute lat und lon definiert worden, welche die Koordinaten eines Punktes auf der Karte angeben. Oder auch:
<Owner b_id="*b0001*">Malte Odenthal</Owner>

Diese Vorgehensweise erstreckt sich über alle XML-Dokumente, da es uns das Erstellen der Schemata vereinfacht hat, denn alle Attribute besitzen *restrictions*.

CacheList – Schema

Das Schema der CacheList wurde von uns so gestaltet, dass alle komplexen Elemente als einzelne *complexTypes* umgesetzt wurde, welche *named* und nicht *anonym* sind, um später eine einfachere Handhabung mit den generierten Klassen zu haben. Aus dem erstellten Schema lässt sich die geplante CacheList gut generieren und das Schema kann so verwendet werden. Bei dem generieren einer XML-Datei aus dem Schema trat ein Problem auf. Die Generierung bestimmter Attribute, welche mit *pattern* und als *base token* definiert sind, erscheint als generierter Inhalt nur „token“ und nicht das gewünschte Ergebnis. Umgehen konnten wir dieses Problem bisher noch nicht. Hier der Code:

```
<xsd:simpleType name="cIdRestriction">
        <xsd:restriction base="xsd:token">
                <xsd:pattern value="[c][0-9]{4}"/> <!-- nur c+4stellige zahl -->
        </xsd:restriction>
</xsd:simpleType>
```

Das Validieren der XML-Datei gegen das Schema funktioniert einwandfrei. Viele *restrictions* sind durch die großen REGEX-Ausdrücke sehr komplex.

User

Die XML-Datei User repräsentiert die Daten eines frisch registrierten User, welche vom Endgerät an den Server verschickt werden, damit dieser die Daten für lange Zeit speichern kann. Hier haben wir uns überlegt, dass es bei der Registrierung durchaus optionale Elemente geben kann, welche wir nachträglich als *optionals* hinzugefügt haben.

```
<optional>

        <news wanted="yes"/>

        <benachrichtigung wanted="yes"/>

        <ortsListe>

                <ort postal="51645" umkreis="10km" lat= "33.12" Lon= "123.12"> Gummersbach-
                Vollmerhausen</ort>

                <ort postal="33501" umkreis="30km" lat="73.12" lon="193.62">

                Teutoburger Wald</ort>

        </ortsListe>

</optional>
```

News und Benachrichtigung sind optional und sind Fragmente der asynchronen Kommunikation. Besonders interessant ist hier die Ortsliste, welche die Orte enthält wo der Benutzer informiert werden soll, wenn dort ein neuer Cache erstellt wurde.
Das Schema-Dokument war sehr einfach, da sehr viele *complexTypes* und *restrictions* sich in allen Dokumenten sehr ähnlich. Sinnvoll wäre es hier z.B. eine Datei mit den mehrfach vorkommenden Elementen zu haben, welche dann über *namespaces* eingebunden wird.

06.05.2013 - Meilenstein: XML-Schemata und Ressourcen

Die Vorstellung unserer XML-Schemata war positiv, uns wurde bestätigt das wir auf einem guten Weg sind und alles, soweit vorauszusehen, so funktionieren müsste.

11.05.2013 - Implementierung des RESTful Webservices

Beim "Representationel state transfer"(REST) handelt es sich um einen Architekturstil, bei dem eine Ressource über eine URI Identifiziert werden kann.
Die Rest Anfrage wird über eine der HTTP-Methoden ausgeführt:

Servicename	entspricht etwa
GET	Read
POST	Create
PUT	Update
DELETE	Delete

Eine Rest Anfrage an einen Server führt eine der oben genannten Methoden aus und ermöglicht die Kommunikation zwischen 2 Geräten (meist Heim-PC an Server).

13.05.2013 - Meilenstein RESTful Webservice

Auch die Abgabe des dritten Meilensteins ist erfolgreich abgearbeitet.

18.05.2013 – erste Schritte in XMPP

Zur Funktionsprüfung der XMPP-Clients wurde ein Openfire-Server Installiert und Konfiguriert, als Libary für die XMPP realisierung nutzen wir die Smack und Smackx APIs.

Erste Testläufe mit der Registrierung von neuen Usern und dem Anmelden eben jener.

23.05.2013 – GUI

Einarbeitung in das Thema GUI mit Java unter Zuhilfenahme der Swing Library.

Serverconnection über GUI realisiert.

25.05.2013 – XMPP

Problem: User braucht Zwei Accounts, einen auf XML und einen auf XMPP Basis

Lösung: User braucht weiterhin Zwei Accounts, beim Registrieren wird auf beiden Seiten ein Identischer Account angelegt, beim Einloggen loggt der User sich auch gleichzeitig auf beiden Seiten ein.

Ausarbeitung und erweitern von XMPP um die Asynchrone Kommunikation zu realisieren.

31.05.2013 – XMPP + GUI

Erweitern unserer XMPP Funktionen.

Erweitern und Verbessern der Benutzeroberfläche.

08.06.2013 – GUI + Implementierung von XML

Begonnen unsere synchronen Daten mit XML in die GUI einzubinden.

12.06.2013 – GUI + Implementierung von XML

Cachefilterung implementiert. Es ist nun möglich Caches anhand einer beliebigen Anzahl an Kriterien zu filtern. Eigene Filterklasse wurde dazu erstellt.

15.06.2013 – GUI + XMPP

Erweitern der GUI um auch die Asynchrone Kommunikation vollständig in die GUI aufzunehmen.

Ausmerzen kleiner noch vorhandener Fehler.

19.06.2013 – XMPP

Problem beim payload -> Fehlersuche

21.06.2013 – Codebaustellen

Weitere kleinere Fehler und nichtmehr benötigte Funktionen aus dem Quellcode entfernen.

22.06.2013 – Dokumentation

Dokumentation

23.06.2013 – Dokumentation + Projektabschluss

Payload problem besteht weiterhin, Daten können nicht angehangen und übertragen werden.

Dokumentation vervollständigt.