2014

Parklotse Online

Dokumentation

Es soll eine Webanwendung entstehen, die die Anzahl freier Parkplätze auf teilnehmenden Parkplätzen anzeigt. Außerdem sollen die Parkplätze auf einer Karte angezeigt werden und Routen empfohlen werden. Zudem kann man Parkgebühren online abrufen. Man kann einen Parkplatz beobachten/abonnieren. Wird dieser Parkplatz nun frei, so erhält man vom System eine Nachricht. Die Applikation soll mithilfe von Sensoren in den Parkplätzen ermitteln können, ob bestimmte Parkplätze noch zur Verfügung stehen. Die Zielgruppe sind die Autofahren die in eine bestimmte Stadt freie Parkplätze suchen. Hierbei soll es den Autofahren möglich sein schneller einen freien Parkplatz zu suchen/bekommen.

Web-basierte Anwendungen 2: Verteilte Systeme
Sommersemester 2014
Prof. Dr. Kristian Fischer
Betreuer: Julian Rahe und Robert Gabriel
Gruppe 2 | Team 22
Arne Beckmann, Daniel Röger und Patrick Rüschenberg
07.05.2014

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

Unsere Aufgabe bestand darin, eine Software zu entwickeln, die Asynchrone, sowie Synchrone Datenübertragung zwischen Server und Client beinhaltet. Zudem sollten wir versuchen uns ein Konzept zu entwickeln, was es bisher nicht gibt. Während unserer Ideensammlung und dem anschließenden Abgleichen der Ideen mit dem Internet, stellte sich heraus, dass alle unsere Ideen nicht neu waren.

Unter anderem wollten wir eine Mitesszentrale entwickeln, in der Leute, die nicht für sich alleine kochen wollen gegen einen Kostenbeitrag zum Essen einladen können.
Dieses System gab es jedoch bereits in dieser Form.

Des Weiteren überlegten wir uns das Konzept einer Web- Anwendung, welche dem User bei der Parkplatzsuche unterstützt. Diese Idee war allerdings auch schon teilweise im Internet vertreten.

Nach Absprache mit unseren Betreuern entschieden wir uns letztere Idee umzusetzen.

1.1. Konzept

Es soll eine Webanwendung entstehen, die die Anzahl freier Parkplätze auf teilnehmenden Parkplätzen anzeigt. Außerdem sollen die Parkplätze auf einer Karte angezeigt werden und die Entfernungen zu diesen angezeigt werden. Zudem kann man Parkgebühren online abrufen. Man kann einen Parkplatz beobachten/ abonnieren. Wird dieser Parkplatz nun frei, so erhält man vom System eine Nachricht.
Die Applikation soll mithilfe von Sensoren in den Parkplätzen ermitteln können, ob bestimmte Parkplätze noch zur Verfügung stehen. Die Zielgruppe sind die Autofahren die in eine bestimmte Stadt freie Parkplätze suchen. Hierbei soll es den Autofahren möglich sein schneller einen freien Parkplatz zu suchen/bekommen.

Wir entschieden uns für dieses Konzept, da man anhand dessen gut asynchrone und synchrone Kommunikation darstellen konnte. Synchrone Kommunikation findet in unserem Konzept immer dann statt, wenn der User Daten vom Server abruft, bzw. Daten hinzufügt. Die asynchrone Kommunikation wird benötigt, damit ein User einen Parkplatz zu dem er sich begeben möchte abonnieren möchte. Hier ist es wenig sinnvoll, wenn er den Quellcode jedes Mal erneut ausführen müsste, um eine Änderung der Parkplatzsituation zu erfahren, da er wahrscheinlich gerade im Auto sitzt.
Damit er diese Informationen trotzdem erhält, entschieden wir uns Aktualisierungen via asynchrone Kommunikation auf seinen Client zu pushen.

Daraus entwickelten wir anschließend eine Systemarchitektur.

2. Architekturelle Fragen

2.1. Systemarchitektur

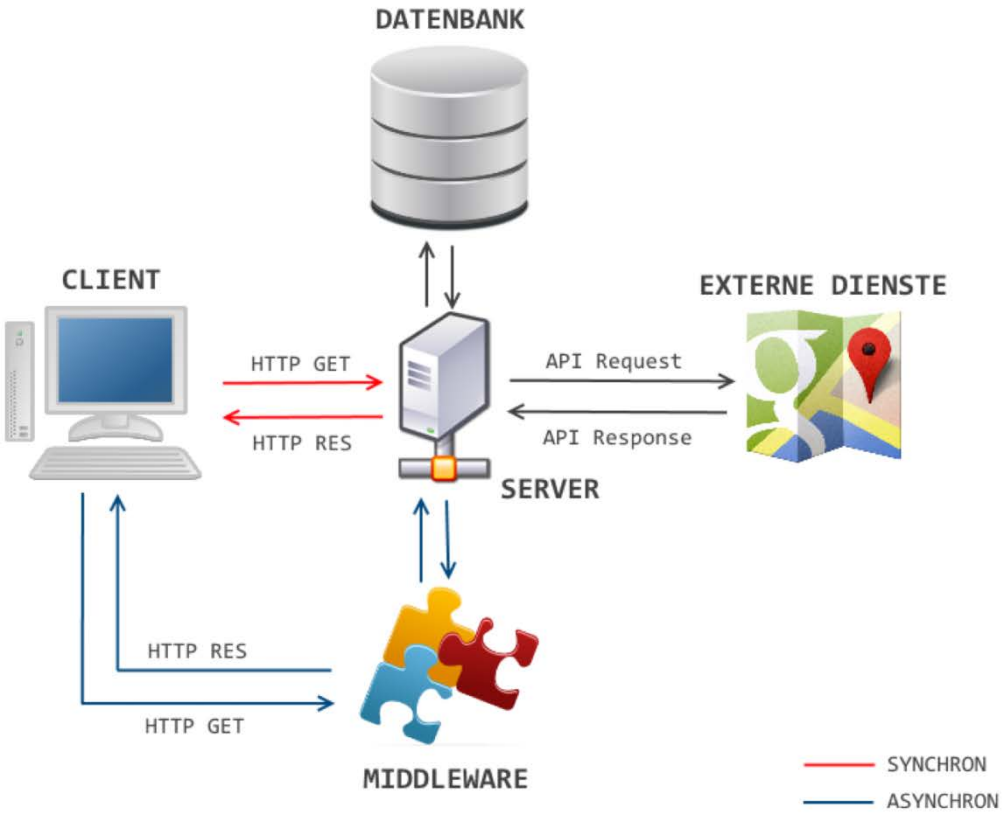

Abbildung 1: Systemarchitektur

Bei diesem Projekt benötigt man einen Topic-Server (Middleware) zwischen dem Web-Server und dem Client um asynchrone Kommunikation zu ermöglichen (Siehe oben). Wenn der User einen Parkplatz abonniert hat, published der Server eine Nachricht, sobald sich der Status des abonnierten Parkplatzes ändert. Diese Nachricht wird als Pushbenachrichtigung an den Client gesendet, solange er auf der Suche nach einem Parkplatz ist.
Alle Informationen zu den aktuellen Parkplätzen und deren Status kann der User synchron vom Server abrufen, der die Datenbank beinhaltet.
Außerdem kommuniziert der Server mit dem externen Dienst Google Map um GPS Standorte auszulesen und wiedergeben zu können. Dies geschieht via API Respone und API Request.

Nachdem wir wussten, wie die Systemarchitektur aussehen sollte überlegten wir konkreter, welche Kommunikationsparadigmen unser System brauchte, um die synchrone und asynchrone Kommunikation umsetzen zu können.

2.2. Kommunikationsparadigmen

Ressourcen

Um das Parkplatzsystem umzusetzen werden nach unseren ersten Überlegungen folgende Ressourcen benötigt:

Ressourcen	HTTP- Methoden
GPS Standort der Parkplätze	GET, POST
Parkplätze	GET, PUT, POST
Parkpreise	GET, PUT, POST
Parkzeiten	GET, PUT, POST
Platzstatus	GET, PUT
User	
Sensor	GET, PUT

Tabelle 1: Ressourcen

Uns erschien die Ressource "GPS Standort" als sinnvoll, da unser System dem User genau anzeigen sollte, welche Parkplätze in seiner Nähe zur Verfügung stehen, da dies das unserer Meinung nach am häufigsten eintretende Szenario war: Eine Person fährt zu einem bestimmten Ort und sucht nun auf dem Weg dorthin, oder nach Erreichen des Ziels einen Parkplatz in der Nähe, der den individuellen Anforderungen entspricht.
Diese Ressource kann der User mit der HTTP-Methode "Get" abrufen. Außerdem erachteten wir die Methode Post für diese Ressource als sinnvoll, damit jederzeit neue Parkplätze, evtl. auch von Privatpersonen zur Verfügung gestellte, ins System aufgenommen werden können.

Unsere nächste Ressource ist "Parkplätze". Hier sind alle Parkplätze enthalten. Diese können mit den Methoden Get, Put, Post abgerufen, verändert und hinzugefügt werden.
Das entspräche in der Realität dann dem Hinzufügen neuer Parkplätze zum System, der Möglichkeit nicht mehr vorhandene Parkplätze zu entfernen. Außerdem kann man zu jedem eingetragenen Parkplatz die Informationen abrufen.
"Parkpreise" und "Parkzeiten" werden ebenfalls festgelegt und können im Nachhinein mit "Get" je ausgelesen werden, mit "Put" verändert werden und mit "Post" hinzugefügt werden.
Dies benötigt man falls sich Preise verändern oder wieder falls neue Parkplätze entstehen bzw. alte wegfallen.

Des Weiteren gibt es die Entitäten "Sensor" und "Platzstatus". Der Sensor wird in unserem System simuliert. In der Realität soll in jedem Stellplatz ein Sensor integriert sein, der anzeigt ob dieser Stellplatz belegt oder Frei ist. Diesen Status findet man in Platzstatus wieder. Sensoren kann man aktualisieren, erzeugen und abrufen. Den Status ebenfalls.

Zusätzlich überlegten wir uns, dass es sinnvoll sein könnte eine Ressource User anzulegen, die laut Vorlesungsstoff auch leer sein konnte.
Im Vortrag des Grobkonzeptes wurde uns davon jedoch abgeraten, bzw. wurde gesagt, dass es sinnvoll sei, "User" HTTP- Methoden zuzuordnen, da man sich mit diesem später anmelden könnte um Parkplätze für diesen "Account" zu abonnieren oder zu reservieren.
Wir entschieden uns im Nachhinein dagegen, da wir keine Cookies und auch keine Sessions verwenden sollten, wodurch man eine Anmeldung des Users ohnehin nur hätte simulieren können (Link auf Seite, die die Sicht eines angemeldeten Users beschreibt).
Die Idee, dass man Parkplätze reservieren könnte lehnten wir ab, da wir uns die Frage stellten: "Was ist, wenn jemand einen Parkplatz befährt, der unser System nicht nutzt. Woher sollte dieser wissen, dass ein Parkplatz reserviert ist, bzw. was sollte ihn davon abhalten dort zu parken?"
Dies zu ermöglichen erschien uns als unrealistisch.

Alle von uns verwendeten Ressourcen werden von unserem Web- Server eingelesen und dem Client des Users von dort aus zur Verfügung gestellt.
Platzstatus und Sensor können hierbei abonniert und damit zusätzlich asynchron abgerufen werden.

Alle Ressourcen stehen dem User synchron zur Verfügung.
Um dies zu realisieren wird folgendes Schemata verwendet:

Schemata

In unserem System benötigen wir neben http keine weiteren Schemata.

Topics

Topics
Land
Bundesland
Stadt
Stadtteil
Einzelner Parkplatz

Tabelle 2: Topics

Nach diesem Schema richtet sich die ursprüngliche Aufteilung der Topics.
Hier hätte der User entscheiden können, in welchem Umfang er einen Parkplatz suchen möchte.
Anfangs hatten wir vorgesehen, dass er hier in der Größenordnung: Stadt, Stadtteil oder Einzelner Parkplatz suchen könnte.
Land und Bundesland wollten wir im Voraus hinzufügen, falls sich das System irgendwann erweitert.

Während wir an unserem Projekt arbeiteten viel uns auf, dass es sinnvoller wäre, wenn dem User beim Start der Applikation Standorte in seiner Nähe angezeigt werden würden.
Anschließend könnte er in einem Suchfeld nach Parkplätzen an anderen Orten suchen.
Hier brächte man die einzelnen Suchumfänge nicht extra einbinden, da man diese einfach mit in das von Google gegebene Suchfeld eingeben könnte. (Bsp. Cityparkplatz in Gummersbach Zentrum)

2.3. Datenstrukturen

JSON Datenstruktur

Unsere JSON Datenstruktur sollte zu Beginn nur aus einem Array bestehen in dem die einzelnen Parkplätze als JSON Objekt gespeichert werden. In diesen Objekten sollten die Adresse sowie die weiteren Informationen zu den Parkplätzen wie z.B. die Maximale Parkdauer gespeichert werden. Ebenfalls sollte der Status, ob ein Parkplatz frei oder belegt ist hinterlegt werden.

Bei der Umsetzung unseres Systems „Parklotse Online“ ist uns aber aufgefallen, dass die Umsetzung mit nur einem Array, in dem die gesamten Parkplätze hinterlegt sind, nicht sehr sinnvoll ist. In diesem Fall würden Parkplätze die zum Beispiel dieselbe Adresse besitzen, doppelte Daten speichern. Da wir die Dopplung von Daten aber vermeiden möchten haben wir uns dazu entschieden, die gesamten Daten auf zwei Arrays aufzuteilen. Diese Arrays werden in der Datenbank von MongoDB als JSON Objekt hinterlegt. Somit haben wir eine Datenstruktur die aus den beiden Arrays „Parkplatz“ und „Stellplatz“ bestehen. Hierbei besteht ein Parkplatz aus einem oder mehreren Stellplätzen und ein Stellplatz kann immer nur genau zu einem Parkplatz gehören. Somit haben wir die Redundanz der Daten behoben.

Daten des JSON Objekt „Parkplatz“

„Parkplatz“	Beschreibung
Parkplatz ID	Eindeutige ID jedes Parkplatzes
Name	Name des Parkplatzes
Stadt	Teil der Adresse – Stadt
PLZ	Teil der Adresse – PLZ
Straße	Teil der Adresse – Straße mit Hausnummer
Preis pro Stunde	Preis pro Stunde Parkdauer in Euro
Preis pro Tag	Preis pro Tag Parkdauer in Euro
Maximale Parkdauer	Die maximal erlaubte Parkdauer des Parkplatzes
Öffnungszeiten	Die Öffnungszeiten wir hier als Code eingetragen. z.B.: 072008181018 Die Ersten vier Ziffern stehen für Montag bis Freitag, die nächsten vier Ziffern für Samstag und die letzten vier für Sonntag. Hierbei stehen die ersten beiden Ziffern des jeweiligen Zeitraumes für die Öffnungszeit und die letzten beiden jeweils für die Schließzeit des Parkplatzes.
Zusatzinformationen	Hier können diverse Zusatzinformationen eingetragen werden.
Typ	Parkplatz oder Parkhaus
Kontakt	Telefonnummer oder E-Mail Adresse des Besitzer/Pächter des Parkplatzes
Breite der Einfahrt	Die maximal erlaubte Breite der Einfahrt
Höhe der Einfahrt	Die maximal erlaubte Höhe der Einfahrt
Breitengrad	Hier werden die Breiten- und Längen- Grade des Parkplatzes eingetragen. Somit ist auch ohne Adresse eine genaue Bestimmung des Parkplatzes möglich.
Längengrad	

Tabelle 3: Daten des JSON Objekt „Parkplatz“

Daten des JSON Objekt „Stellplatz“

„Stellplatz“	Beschreibung
Stellplatz ID	Eindeutige ID jedes Stellplatzes
Parkplatz ID	Hier wird die ID des Parkplatzes eingetragen zu dem der Stellplatz gehört. Maximal eine ID möglich.
Status des Sensors	Im Status des Sensors wird hinterlegt ob der Stellplatz frei oder belegt ist. Ebenfalls lässt sich mit dem Status die Anzahl der freien und belegten Stellplätze eines Parkplatzes berechnen.
Frauenparkplatz	Zusatzinformation - Frauenparkplatz
Behindertenparkplatz	Zusatzinformation - Behindertenparkplatz
Eltern Kind Parkplatz	Zusatzinformation - Eltern Kind Parkplatz

Tabelle 4: Daten des JSON Objekt „Stellplatz“

Beispiel des JSON Objekt „Parkplatz“

```
var Parkplatz = [
        # erster Parkplatz
        {
            "Parkplatz_ID": 0,
            "Name": "FH Parkplatz",
            "Stadt": "Gummersbach",
            “PLZ”: 51643,
            "Straße": "Steinmüllerallee 1",
            "Preis_pro_Stunde": 0.5,
            "Preis_pro_Tag": 0.5,
            "Maximal_Parkdauer": 2,
            “Oeffnungszeiten”: 002400240024,
            "Zusatzinfo": “Geschlossen”,
            "Typ": “Parkhaus”,
            "Kontakt": “email@email.de”,
            “Breite”: 3,
            “Höhe”: 2,
            “LON”: 51028879,
            “LAT”: 07567497
        },
        # zweiter Parkplatz
        {
                        ...
        }
]
```

Abbildung 2: Beispiel des JSON Objekt „Parkplatz“

Beispiel des JSON Objekt „Stellplatz“

```
var Stellplatz =[
    # erster Stellplatz auf Parkplatz mit der ID 0
    {
            "Stellplatz_ID": 0,
            "Parkplatz_ID": 0,
            "Status_Frei": true,
            "Freuenparkplatz": false,
            "Behindertenparkplatz": false,
            "Eltern_Kind_Parkplatz": false
    },
    # zweiter Stellplatz
    {
                ...
    }
]
```

Abbildung 3: Beispiel des JSON Objekt „Stellplatz“

2.4. Kommunikationsmodell

Unser Kommunikationsmodell zeigt die synchrone und asynchrone Kommunikation zwischen dem Parkplatz suchendem (Autofahrer) und unserem Parkplatz System.

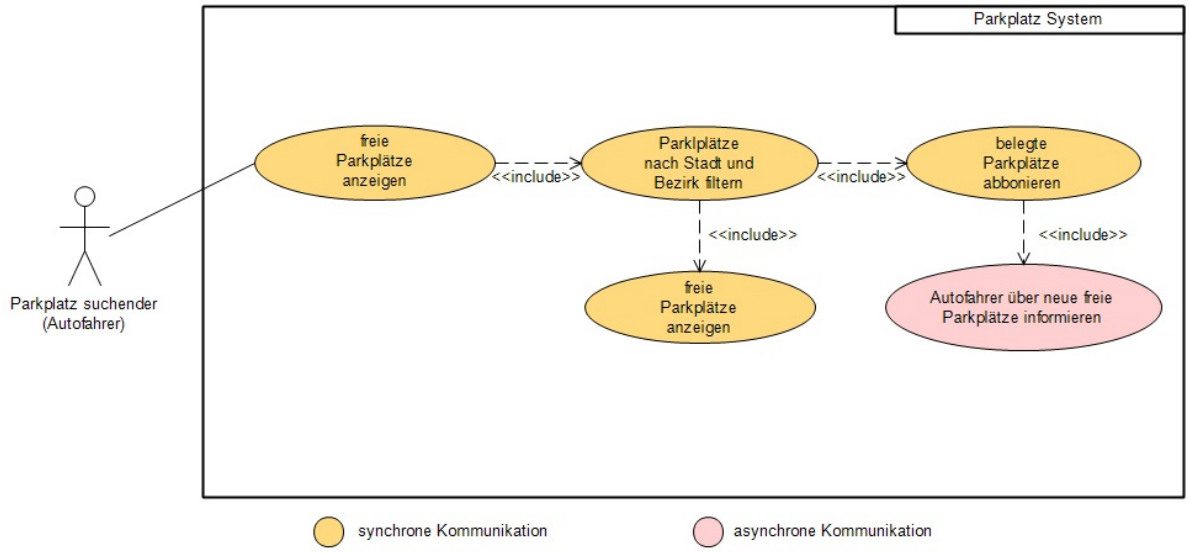

Abbildung 4: Kommunikationsmodell

Unser Parkplatz System Modell aus unserer Konzeptionellen Phase sollte es einem Parkplatz suchendem Autofahrer ermöglichen in seiner näheren Umgebung freie Parkplätze zu finden. Diese Freien Parkplätze sollte der Autofahrer nach unserem Topic Filter können umso noch eine genauere Auswahl der Parkplätze zu bekommen. Belegte Parkplätze sollte der Autofahrer abonnieren können und bei frei werden eine Rückmeldung bekommen. Hierbei sollte das anzeigen, filtern und abonnieren der Parkplätze mit synchroner Kommunikation funktionieren und die update Funktion, die den Autofahrer über neu frei gewordene Parkplätze informiert, mit der Asynchronen Kommunikation.

Bei der Umsetzung hat sich unseres Systems ein wenig verändert. Nun bekommt der Anwender beim Aufrufen der Seite direkt eine Übersicht über alle Parkplätze die sich in seiner näheren Umgebung befinden. Hierbei ist die Entfernung auf zwei Kilometer Luftlinie eingestellt. Auf der Startseite hat der Anwender ebenfalls die Möglichkeit unter der Suchfunktion eine Adresse einzugeben. Hierbei bekommt man dann, die in der Nähe der eingegebenen Adresse befindenden Parkplätze angezeigt.
Sobald die Startseite aufgerufen oder eine Suche gestartet wird abonniert man die Parkplätze die sich in der Nähe befinden.
Über die Asynchrone Kommunikation wird der Anwender immer auf den neusten Stand gehalten, wie viele Parkplätze belegt und frei sind.

3. Prozessdarlegung

3.1. Umsetzung/Probleme

Allgemeines Design

Für unser Design haben wir uns zu Beginn ein dynamisches Design vorgestellt, was sich in jedem Browser dynamisch an die Breite und Höhe anpasst. Unsere Idee hierbei war dass sich unser System an jede Auflösung anpasst. Hiermit sollten die Smartphone abgedeckt werden. Bei der Programmierung des Designs hatten wir anfangs Probleme die Auflösung für Smartphone richtig anzupassen.
Nach einigem experimentieren haben wir es dann doch einbinden können.

Logo

Um unserer Applikation einen Wiedererkennungswert mitzugeben haben wir uns überlegt.
Dieses enthält das offizielle Parkhaussymbol der deutschen StVO, welches wir um den Namen der Anwendung erweitert haben.
Wir entschieden uns für dieses Logo, da man hier direkt erkennt, was sich ungefähr dahinter verbirgt.
Die Farben des Frontends haben wir an die Farbe des Logos bzw. des Straßenschildes angepasst um ein einheitliches Design zu erhalten.
Wir haben uns dazu entschieden kein Backend zu erstellen, da wir dies zeitlich nicht mehr hätten ausführen können und wir den Hintergedanken hatten, dass man dies nicht benötigt um asynchrone und synchrone Kommunikation darzustellen.

Abbildung 5: Logo Parklotse - Online

Datenstruktur

Nach der Planung in unserem Grobkonzept hatten wir nur eine JSON Datenstruktur. Nach einer längeren Diskussion über unsere Datenstruktur sind wir zu dem Ergebnis gekommen zwei Datenstrukturen zu verwenden. Mit unserer Idee aus dem Grobkonzept wäre es schwieriger geworden einzelne Parkplätze, die sich an einer Straße befinden, Einzugliedern und diese in unserem System zu suchen und zu filtern. Mit zwei Datenstrukturen wo einzelne Stellplätze einem Parkplatz zugeordnet werden können, wird die Zuordnung auch an der Straße möglich.

Anonym oder mit User Login

Bei der Überlegung ob wir einen Login Bereich brauchen, sind wir nach einer Diskussion zu dem Ergebnisse gekommen, das unser System auch mit Anonymen Benutzern funktioniert und wir somit erstmals keinen Login benötigen. Zuerst überlegten wir diese noch nachzurüsten, falls wir am Ende unseres Projektes noch Zeit hätten.

Anschließend wurde uns noch mitgeteilt, dass man weder Cookies, noch Sessions verwenden sollte und somit ein Login System nicht realisierbar sei. Man hätte ein solches höchstens durch Verlinkung mehrerer Seiten simulieren können.
Aufgrund dessen haben wir am Ende ganz auf das Login System verzichtet. (Siehe auch Kommunikationsparadigmen/User)

Google API

Da es die Google API nicht ermöglicht per Ajax xss auf Json Ressourcen anzuwenden entschieden wir uns Mapinhalt per iframe einzubinden.

Simulation eines Sensors

Da unser System auf Sensoren, die in jedem Parkplatz eingebaut sind basiert, mussten wir diese Sensoren simulieren.
Diese Simulation realisierten wir auf unsere Server mithilfe von Javascript. (Siehe Abbildung Unterhalb)

Das Programm für den Sensor durchläuft die gegebene maximale Parkplatzanzahl und zählt die Anzahl der freien Plätze.
Am Ende des Sensor- Codes wird mit zwei if- Abfragen sichergestellt, dass die Summe der belegten und freien Parkplätze innerhalb eines Parkhauses die maximale Parkplatzanzahl nicht überschreiten kann und dass deren Differenz die minimale Parkplatzanzahl nicht unterschreiten kann.
Danach wird die Anzahl der belegten Parkplätze übermittelt.

```
app.get('/randsensor', function(req, res){
        //Amount of Parkinglots
        var counter;
        var max = new Array();
        parkinglots.findItems(function (error, result){
            result.forEach(function(parkinglots){
                counter++;
                parkingspace.findItems(function(error, result2){
                    var counter2 = 0;
                    var counter3 = 0;
                    result2.forEach(function(parkingspaces){
                        if(parkingspaces.parkplatz_id == parkinglots.id){
                            counter2++;

                            if(parkingspaces.status == "frei"){
                                counter3++;
                            }
                        }
                    });
                    var rand_plusminus = Math.floor((Math.random() *2));
                    var rand_amount = Math.floor((Math.random() * 5));

                    if(rand_plusminus == 0 && counter3+rand_amount <= counter2){
                        var publication = pubClient.publish('/parkingslots/'+parkinglots.id, {
                            "amount": counter3+rand_amount

                        });
                    }else if(rand_plusminus == 1 && counter3-rand_amount >= 0){
                        var amount = counter3-rand_amount;
                        var publication = pubClient.publish('/parkinglots/'+parkinglots.id, {
                            "amount": amount

                        });
                    }
                });
            });
        });

```

Abbildung 6: Simulation eines Sensors

3.2. Zeitplan

Datum	Ort	Mitglieder	Aufgabe
Bis zum 08.04.2014	FH GM	Arne, Daniel, Patrick	Erarbeiten der zuvor gestellten Übungsaufgaben, Exposé und Grobkonzeptes
09.04.2014	FH GM	Arne, Daniel, Patrick	Vorstellung des Grobkonzeptes (3 Seiten Präsentation)
10.04.2014	FH GM	Daniel, Patrick	Diskussion über die Datenstruktur und das Problem einzelner Parkplatz Stellplätze
11.04.2014	FH GM	Arne, Daniel, Patrick	Beginn der Programmierung des Servers und einer Grundlegenden HTML Seite
14.04.2014	FH GM	Arne, Patrick	Programmierung der Standortermittlung über den Browser
16.04.2014	FH GM	Arne, Daniel, Patrick	Festlegen der Endgültigen Datenstruktur und Umsetzung dieser in Mongo DB
23.04.2014	FH Köln	Arne, Daniel, Patrick	Weiter Programmierung der Standortermittlung über den Browser und Beginn mit der Ausgabe der Parkplätze in der Nähe des Aktuellen Standortes
24.04.2104	Skype	Arne, Daniel	Kurze Zwischenstandbesprechung was erledigt und was noch Zutun ist
25.04.2014	FH Köln	Arne, Daniel, Patrick	Weitere Programmierung des Ausgabe im Browser
28.04.2014	FH GM	Arne, Daniel, Patrick	Dokumentation
29.04.2014	FH GM	Arne, Daniel, Patrick	Besprechung des bisherigen Standes und Planung der letzten Phase des Projektes
30.04.2014	FH GM	Arne, Daniel, Patrick	Zwischen Stand Vorstellen und mit Betreuern besprechen
02.05.2014	FH GM	Arne, Daniel, Patrick	Arbeiten am Code und der Dokumentation
04.05.2014	Skype	Arne, Daniel, Patrick	Zusammentragen der Neuerungen des Codes, Erweiterung der Dokumentation, Verabredung des neuen Termins
05.05.2014	FH GM	Arne, Daniel, Patrick	Gemeinsame Arbeit am Code und der Dokumentation, Hochladen der finalen Version unsres Systems, Besprechung was noch Zutun ist
06.05.2014	FH GM, Skype	Arne, Daniel, Patrick	Arbeiten und Fertigstellen der Dokumentation, Vorbereitung für die Abgabe und Vorstellung des Projektes
07.05.2014	FH GM	Arne, Daniel, Patrick	Abgabe (bis 8Uhr) und Vorstellung des Projektes (um 12:45-13:00 Uhr)

Tabelle 5: Zeitplan

3.3. Arbeitsmatrix

Thema	Arne	Daniel	Patrick
Konzept/Idee	33%	33%	33%
Grobkonzept	33%	33%	33%
Dokumentation	33%	33%	33%
Programmierung	33%	33%	33%
Präsentation	33%	33%	33%
Organisation	33%	33%	33%

Tabelle 6: Arbeitsmatrix

4. Fazit

Abschließend kann man sagen, dass die Arbeit und Organisation innerhalb unserer Gruppe sehr gut geklappt hat.
Zwar hatten wir zu Beginn des Projektes Probleme ein gutes Konzept zu entwickeln und uns in das in den Workshops vermittelte Wissen einzuarbeiten, am Ende kamen wir jedoch zurecht.
Das Modul war sehr Zeitaufwändig, dennoch haben wir es geschafft uns häufig zusammenzusetzen oder weiter Vorgehensweisen zu Kommunizieren und somit eine angemessene Workload zu investieren..
Bei der Umsetzung von Parklotse- Online stießen wir teilweise auf Probleme. Hier mussten wir teilweise von unserer ursprünglichen Planung abweichen und eine neue Lösung finden. So hielten wir die Ressource des Users, die eine Anmeldung und Profile ermöglicht hätte bei der Umsetzung nicht mehr für notwendig.
Ähnlich verhielt es sich mit Topics, die der User nach erster Planung hätte auswählen können um seine Suche einzugrenzen. Diese kann der User im Endergebnis nichtmehr per Click auswählen, da sie automatisch in der Google API vorhanden sind, wenn man beispielsweise die Stadt mit in das Suchfeld eingibt.
Alles in allem konnten wir aber die meisten Punkte aus dem Grobkonzept einhalten und sind zufrieden mit unserem Ergebnis.

5. Anhang

5.1. Exposé

Das Konzept

Es soll eine Webanwendung entstehen, die die Anzahl freier Parkplätze auf teilnehmenden Parkplätzen anzeigt. Außerdem sollen die Parkplätze auf einer Karte angezeigt werden und Routen empfohlen werden. Zudem kann man Parkgebühren online abrufen. Man kann einen Parkplatz beobachten/ abonnieren. Wird dieser Parkplatz nun frei, so erhält man vom System eine Nachricht.
Die Applikation soll mithilfe von Sensoren in den Parkplätzen ermitteln können, ob bestimmte Parkplätze noch zur Verfügung stehen.

Die Zielgruppe

Die Zielgruppe sind die Autofahren die in eine bestimmte Stadt freie Parkplätze suchen. Hierbei soll es den Autofahren möglich sein schneller einen freien Parkplatz zu suchen/bekommen.

Die Kommunikationsparadigmen

Die Applikation beruht auf synchronen und asynchronen Datenübertragungsvorgängen. Synchron würde hier beispielsweise die Route übertragen, die zum nächsten Parkplatz führt. Asynchrone Datenübertragung findet man bei push Notifikation, die erscheinen, sobald auf einem beobachteten Parkplatz wieder ein Platz frei wird.

Marktrecherche

Es gibt bereits ein Pilotprojekt.
http://www.iphone-ticker.de/video-braunschweig-testet-iphone-gestutztes-parkplatzsystem-42608/

5.2. Grobkonzept

Systemarchitektur

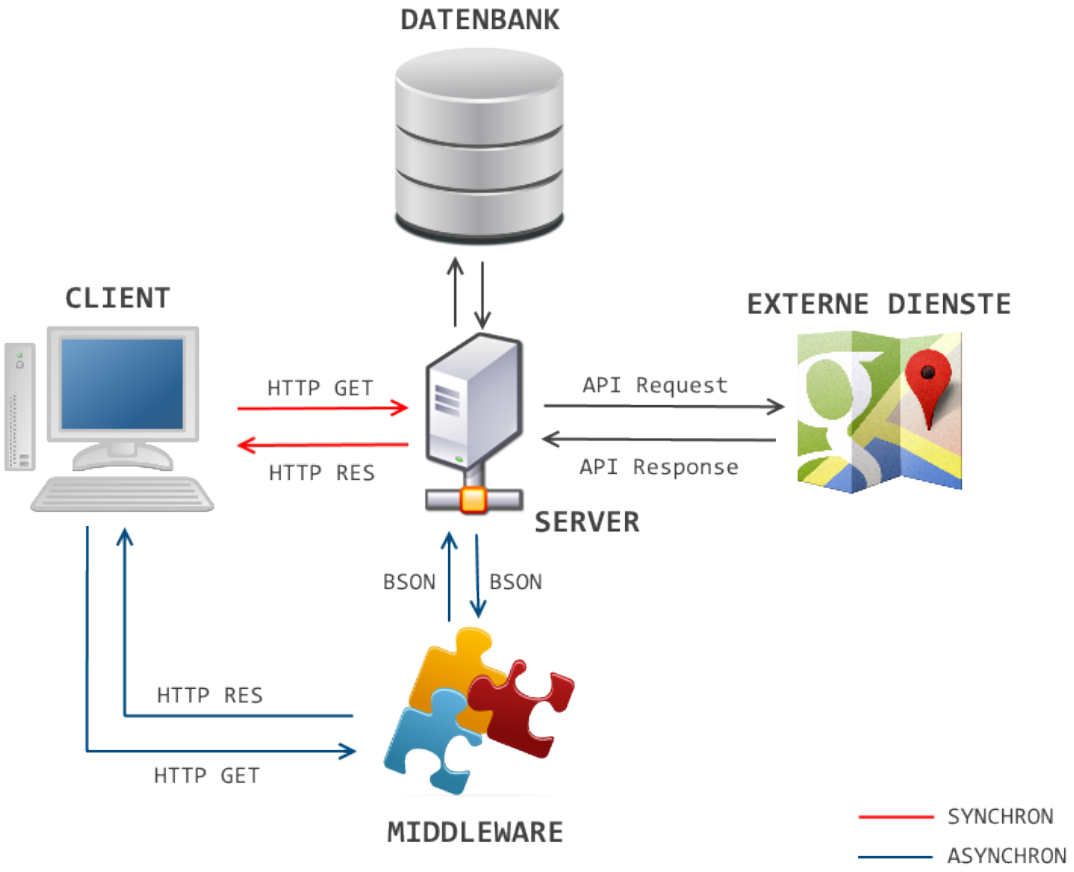

Abbildung 7: Grobkonzept Systemarchitektur

Kommunikationsparadigmen

Ressourcen

Um das Parkplatzsystem umzusetzen werden folgende **Ressourcen** benötigt:

Ressourcen	HTTP- Methoden
GPS Standort der Parkplätze	GET, POST
Parkplätze	GET, PUT, POST
Parkpreise	GET, PUT, POST
Parkzeiten	GET, PUT, POST
Platzstatus	GET, PUT
User	
Sensor	GET, PUT

Tabelle 7: Grobkonzept Ressourcen

Sie werden von unserem Web- Server eingelesen und dem Client des Users von dort aus zur Verfügung gestellt.
Platzstatus und Sensor können hierbei abonniert und damit zusätzlich asynchron abgerufen werden.
Alle Ressourcen stehen dem User synchron zur Verfügung.
Um dies zu realisieren wird folgendes Schemata verwendet:

Schemata

Schemata
http

Tabelle 8: Grobkonzept Schemata

Topics

Topics
Land
Bundesland
Stadt
Stadtteil
Einzelner Parkplatz

Tabelle 9: Grobkonzept Topics

Nach diesem Schema richtet sich die Aufteilung der Topics. Hier kann der User entscheiden, in welchem Umfang er einen Parkplatz suchen möchte.
Er kann hier in der Größenordnung: Stadt, Stadtteil oder Einzelner Parkplatz suchen.
Land und Bundesland werden voraussehend hinzugefügt, falls sich das System irgendwann erweitert.
Bei diesem Projekt benötigt man einen Topic- Server zwischen dem Web- Server und dem Client. Wenn der User einen Parkplatz abonniert hat, published der Server eine Nachricht, sobald sich der Status des abonnierten Parkplatzes ändert. Diese Nachricht wird als Pushbenachrichtigung an den Client gesendet, solange er auf der Suche nach einem Parkplatz ist.

Datenstrukturen

JSON Datenstruktur

Unsere JSON Datenstruktur besteht aus einem Array in dem die einzelnen Parkplatze als JSON Objekt gespeichert werden. In diesen Objekten werden die Adresse sowie die weiteren Informationen zu den Parkplätzen wie z.B. die Maximale Parkdauer gespeichert. Hier wird ebenfalls der Status, ob ein Parkplatz frei oder belegt ist hinterlegt.

JSON Beispiel

```
var Parkplatz = [
        # erster Parkplatz
        {
                "Parkplatz_ID": 0,
                "Land": "Deutschland",
                "Bundesland": "NRW",
                "Stadt": "Gummersbach",
                "Bezirk": "Zentrum",
                "Straße": "Steinmüllerallee",
                "Nr.":0,
                "Gebuehren_in_Euro_pro_Stunde": 0.5,
                "Maximal_Parkdauer_in_Stunden": 2,
                "Status_Frei": true,
                "Freuenparkplatz": false,
                "Behindertenparkplatz": false,
                "Eltern-Kind_Parkplatz": false
        },
        # zweiter Parkplatz
        {
                ...
        }
]
```

Kommunikationsmodell

Unser Kommunikationsmodell zeigt die synchrone und asynchrone Kommunikation zwischen dem Parkplatz suchendem (Autofahrer) und unserem Parkplatz System.

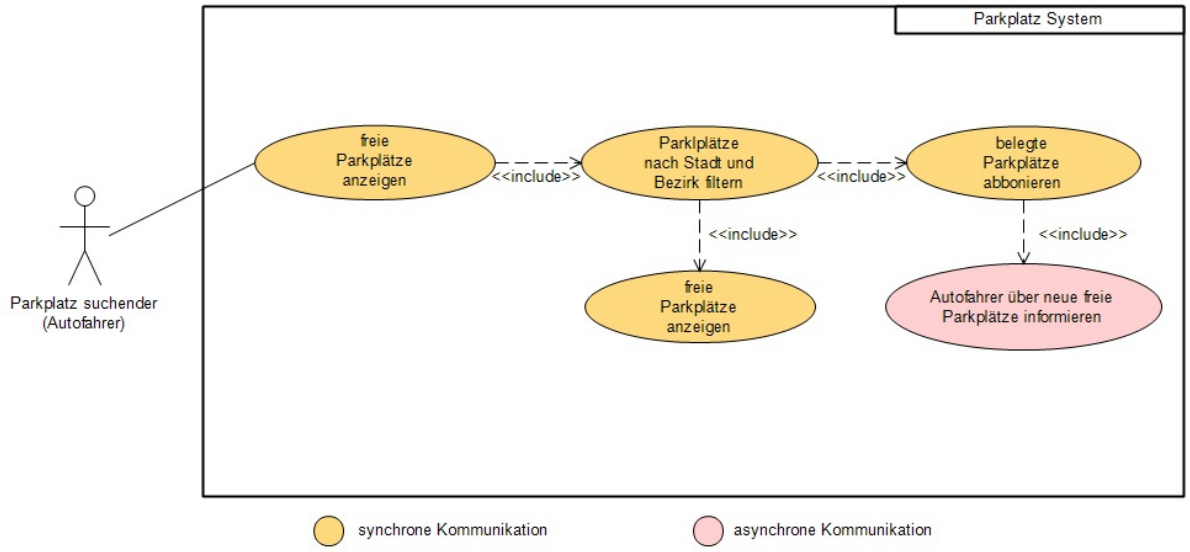

Abbildung 8: Grobkonzept Kommunikationsmodell

5.3. Planung der Meilensteine

Meilensteine

09.04. Grobkonzept

23.04. Userinterface, Grundfunktionen des Servers(synchrone, asynchrone Kommunikation)

27.04. Fertigstellung Zwischenstand

30.04. Zwischenstand

04.05. Fertigstellung

07.05. Abgabe des Projekts

Herausforderung

Zwischenziele termingerecht einhalten

Vorlesungsinhalte abdecken

Projekt in der richtigen Größenordnung halten

Ausreichende und verständliche Kommunikation innerhalb des Teams