Bachelorarbeit

Bastian Bliemeister

GeoTracking von Webseiten-Zugriffen
-
Eine prototypische Entwicklung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Bastian Bliemeister

GeoTracking von Webseiten-Zugriffen
-
Eine prototypische Entwicklung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. sc. pol. Wolfgang Gerken
Zweitgutachter : Prof. Dr. Thomas Thiel-Clemen

Abgegeben am 08. November 2010

Bastian Bliemeister

Thema der Bachelorarbeit

GeoTracking von Webseitenzugriffen und deren Darstellung auf einer geografischen Karte in Echtzeit.

Stichworte

Web-Analytics, Web-Controlling, GeoTracking

Kurzzusammenfassung

Diese Bachelorarbeit befasst sich mit dem GeoTracking von Webseiten-Zugriffen. Die Zugriffe eines Benutzers auf einer Webseite werden in einer Sitzung zusammengefasst und in Echtzeit auf einer geografischen Karte dargestellt. Bei neuen oder abgelaufenen Sitzungen wird die Karte automatisch aktualisiert und zeigt damit immer die Standorte der aktiven Benutzer einer Webseite in Echtzeit an.

Komponenten des Systems sind unter anderem Oracle 11g, Oracle Application Express, Oracle MapViewer, Apache Tomcat und Apache HTTP-Server.

Bastian Bliemeister

Title of the paper

GeoTracking of website requests and their representation on a map in real time.

Keywords

Web-Analytics, Web-Controlling, GeoTracking

Abstract

This paper deals with GeoTracking of website requests. The Requests of a user on a website are summed up in one session and are displayed in real time on a map. The map is updated automatically when new sessions are created or if the sessions expire. Therefore the location of active user on a website is always reflected in real time.

Components of the system include Oracle 11g, Oracle Application Express, Oracle MapViewer, Apache Tomcat and Apache HTTP-Server.

Inhaltsverzeichnis

1 Einleitung....5
2 Web-Analytics....7
2.1 Allgemeines....7
2.1.1 Definition....7
2.1.2 Einsatzgebiete....7
2.2 Verfahren....7
2.2.1 Webserver-Logdateianalyse....8
2.2.2 Tags und Pixel....9
2.2.3 Cookies....9
2.2.4 Third-Party-Cookies....10
2.2.5 Pro & Contra....10
2.2.6 Kombination unterschiedlicher Verfahren....11
2.3 Methoden....11
2.3.1 Monitoring....11
2.3.2 Optimierungen....12
2.4 Datenschutz....12
2.5 Fazit....12
3 Webserver....14
3.1 Apache HTTP-Server....14
3.2 Apache Tomcat....14
3.3 Microsoft IIS....15
3.4 Lighttpd Webserver....15
3.5 Pro & Contra....15
3.6 Fazit....16
4 Pixel-Anwendung....18
4.1 Design der Anwendung....18
4.2 Design eines Moduls....19
4.3 Interaktion der Module mit der Anwendung....20
4.4 Extract Transform Load Modul....21
4.4.1 GeoCity Datenbank....21
4.4.2 Extract....22
4.4.3 Transform....23
4.4.4 Load....25
4.5 Pixel-Servlet....27
4.6 Einstellungen: web.xml und context.xml....28
4.7 Probleme und Lösungen....29
5 Datenbankdesign....30
5.1 Views....31
6 Kartendarstellung....33
6.1 Kartenmaterial und Geodaten....33
6.1.1 NAVTEQ....33
6.1.2 Google Maps....33
6.1.3 OpenStreetMaps....34
6.2 Darstellung des Kartenmaterials....34
6.2.1 NAVTEQ....34
6.2.2 Google Maps....34
6.2.3 OpenStreetMaps....35
6.3 Web Map Service....35

6.4 Vorteile und Nachteile....36
6.5 Oracle MapViewer und Oracle MAPS....37
6.6 Fazit....37
6.7 Zusammenspiel der Komponenten....38
7 Oracle Application Express....40
8 Komplettes System....42
8.1 Konfiguration: Oracle 11g....42
8.2 Konfiguration: Oracle APEX....42
8.3 Konfiguration: Oracle MapViewer....42
8.4 Konfiguration: Apache Tomcat....45
8.5 Konfiguration: Apache HTTP-Server....45
8.6 APEX-Anwendung....48
8.6.1 Einbinden der Karte....48
8.6.2 Anzeige der Markierungen....48
9 Aktualisierung in Echtzeit....50
10 Fazit....54
Literaturverzeichnis....57
Tabellenverzeichnis....58
Abbildungsverzeichnis....58

1 Einleitung

In dieser Bachelorarbeit wird ein System erstellt welches Zugriffe auf eine Webseite auswertet, den Aufrufen den konkreten Standort des Aufrufes zuordnet und diese in Echtzeit auf einer geografischen Karte grafisch darstellt. Vorgabe ist es die erhobenen Daten in einer Oracle-Datenbank zu speichern und die Webseite, auf der die Karte dargestellt wird, mit dem Oracle Application Express zu generieren.

Die Webzugriffe sollen in Echtzeit auf der Karte dargestellt werden. Ein Webzugriff soll also direkt nach dem Auswerten und dem Schreiben in die Datenbank auf der Karte erscheinen. Hierfür ist es erforderlich, dass die Webseite, die die Karte darstellt über neue Zugriffe informiert wird, um eine Aktualisierung der Karte durchzuführen.

Weiterhin soll das gesamte System skalierbar sein, um es an steigende Lastaufkommen anpassen zu können und auf Windows und Linux-Servern gleichermaßen lauffähig sein.

In der Ausarbeitung werde ich mich intensiv mit den einzelnen Komponenten des Systems und den nötigen Verfahren auseinandersetzen. Folgende Komponenten werden zur Realisierung benötigt:

- Einen Webserver, im weiteren 'Pixel-Server' genannt, der die Ressourcen bereithält die von den Aufrufern angefordert werden.
- Eine Anwendung, im weiteren 'Pixel-Anwendung' genannt, die auf dem Pixel-Server läuft und die einzelnen Aufrufe auswertet, die benötigten Informationen extrahiert und persistent speichert.
- Eine Datenbank in der die extrahierten Daten persistent gespeichert werden.
- Einen Webserver von dem die ermittelten Daten auf einer Karte dargestellt werden.

Zu Beginn beschäftige ich mich mit verschiedenen Techniken der Web-Analytics und wähle eine geeignete Kombination um die Aufgabenstellung zu erfüllen.

Ich werde verschiedene Webserver vorstellen und vergleichen, um einen geeigneten Webserver für die 'Extract Transform Load'-Komponente (ETL) zu finden. Innerhalb dieser Komponente werden die Daten aus den Webseitenzugriffen extrahiert, aufbereitet und in eine Datenbank geschrieben. Für diese Aufgabe habe ich eine Pixel-Anwendung entworfen, die ich anschließend vorstellen werde. In diesem Kapitel gehe ich auch auf die Datenbank ein, mit der die erforderlichen Geodaten über den Standort des Aufrufers ermittelt werden.

Als Datenbank zur Speicherung der Zugriffe kommt, laut Aufgabenstellung, eine Oracle-Datenbank zum Einsatz. Ich werde das entworfene Datenbankdesign, welches für die Speicherung der Zugriffe notwendig ist, im Detail vorstellen.

Weiterhin vergleiche ich verschiedene Anbieter von Kartenmaterial und die dazugehörigen Programme und Techniken, die zum Bereitstellen und Darstellung der Karte auf einer Webseite benötigt werden. Anhand der Vorteile und Nachteile der einzelnen Anbieter und Techniken werde ich mich für eine geeignete Lösung entscheiden.

Die Generierung der Webseite erfolgt, laut Aufgabenstellung, über den Oracle Application Express (APEX). APEX läuft innerhalb der Oracle-Datenbank und erlaubt es schnell und unkompliziert neue Auswertungen der gesammelten Daten zu erstellen. Als Webserver für die APEX-Anwendung kommt der Oracle PL/SQL-Server zum Einsatz.

Am Ende der Ausarbeitung werde ich alle Komponenten entsprechend konfigurieren, zusammenführen und auf die Technik eingehen, mit der die Karte in Echtzeit aktualisiert wird. Anschließend ziehe ich mein Fazit.

2 Web-Analytics

In diesem Kapitel beschäftige ich mich mit der Web-Analytics im Allgemeinen aber auch mit den verschiedenen Verfahren und Methoden der Web-Analytics.

Zu Beginn gehe ich auf die Web-Analytics bzw. wo sich die Einsatzgebiete befinden ein. Nach der Betrachtung der einzelnen Verfahren und Methoden werde ich am Ende des Kapitels das Thema Datenschutz anreißen und mich im Fazit für eine geeignete Kombination der gezeigten Verfahren entscheiden.

2.1 Allgemeines

Für Web-Analytics wird in Deutschland oft der Begriff Web-Controlling verwendet, ist aber auch unter vielen weiteren Bezeichnungen bekannt. In meiner Ausarbeitung werde ich ausschließlich den Begriff 'Web-Analytics' verwenden.

Web-Analytics ist für professionelle Internetauftritte von Unternehmen essentiell. Ein Webauftritt kostet bei der Erstellung und Wartung viel Geld und soll in der Regel einen Gewinn für das Unternehmen darstellen. Eine effiziente und optimierte Webpräsenz ist aus diesem Grund eine Investitionssicherung für ein Unternehmen. Dies zu erreichen ist die Aufgabe der Web-Analytics.

2.1.1 Definition

Web Analytics (auch Web Controlling, Web-Analyse, Datenverkehrsanalyse, Traffic-Analyse, Clickstream-Analyse, Webtracking) ist die Sammlung und Auswertung des Verhaltens von Besuchern auf Websites. [1]

2.1.2 Einsatzgebiete

Die Web-Analytics dient der Überwachung eines Webauftrittes, um diesen zu optimieren und effizienter zu gestalten. An dem Zweck des Webauftritts orientiert sich die Analyse.

Bei einem Webshop werden beispielsweise Statistiken über die Warenkörbe benötigt. Welche Produkte werden einem Warenkorb hinzugefügt, welche Produkte entfernt, welche Warenkörbe werden bestellt und welche Warenkörbe verfallen ohne Bestellung. Alles interessante Informationen für eine Optimierung.

Besonders die Analyse der Standorte der Benutzer ist für alle Sparten von Webauftritten interessant. Diese Analysen können Aufschluss darüber geben, wo ein Webauftritt bekannt oder eher unbekannt ist. Mit diesem Wissen können gezielt örtliche Kampagnen geplant werden, die die Bekanntheit des Webauftrittes lokal steigern. Ein Unternehmen kann aus diesen Analysen aber auch Erkenntnisse für neue Niederlassungen gewinnen, in dem es aus den Daten schließt, wo sich die meisten potenziellen Kunden befinden und wie weit diese von einem geplanten neuen Standort entfernt sind.

2.2 Verfahren

In diesem Kapitel stelle ich die am häufigsten, in der Praxis, eingesetzten Verfahren der Web-Analytics vor und führe die Pro & Contra der einzelnen Verfahren auf um anschließend aufzuzeigen welche Verfahren sich miteinander verbinden lassen.

2.2.1 Webserver-Logdateianalyse

Bei der Logdateianalyse wird sich zu nutzen gemacht, dass ein Webserver (Apache Server, Apache Tomcat, MS IIS usw.), auf dem die zu überwachende Website betrieben wird, üblicherweise alle Zugriffe in einer Protokolldatei vermerkt. Diese sogenannte Logdatei, die eigentlich zur Fehlerbehebung gedacht ist, wird von geeigneter Software mittels regulären Ausdrücken ausgewertet. Es werden aus dieser Datei Statistiken erstellt und gegebenenfalls mit Grafiken und Tabellen visualisiert. Alternativ oder ergänzend kann auch auf die Logdateien von Firewalls oder z.B. zwischengeschalteten Proxys zurückgegriffen werden.

Webserver benutzen für die Einträge in der Logdatei im allgemeinen das Common Log Format (CLF). So ist sichergestellt, dass nicht für jeden Webserver ein eigener Parser genutzt werden muss um die Daten zu extrahieren.

Ein Beispiel für einen Logeintrag wäre:

```
IP-Adresse – Userid [Zeitpunkt]
"Methode Ressource Protokoll" Status-Code Größe

127.0.0.1 - user [10/Oct/2000:13:55:36 -0700]
"GET /apache_pb.gif HTTP/1.0" 200 2326
```

IP-Adresse
: Die IP-Adresse des Aufrufers.

Userid
: HTTP-Authentifikations Benutzer-ID

Zeitpunkt
: Zeitpunkt an dem die Ressource aufgerufen wurde.

Methode
: Die Methode mit der die Ressource aufgerufen wurde.

Ressource
: Die Ressource die aufgerufen wurde.

Protokoll
: Das Protokoll mit dem die Ressource aufgerufen wurde.

Status-Code
: Der Status-Code der Übertragung (z.B. `200 → OK`).

Größe
: Die Größe der übertragenen Daten in Bytes ohne Response-Header.

Mittels der protokollierten IP-Adresse und aufgerufenen Seite ist es einfach möglich, ein Bewegungsprofil in chronologischer Abfolge über den gesamten Besuch eines Benutzers auf der Webseite zu erstellen. Unter anderem ist es auch möglich, die Anzahl der Seitenaufrufe der einzelnen Seiten einer gesamten Webseite zu ermitteln und daraus auf deren Beliebtheit zu schließen.

Mit der Logdateianalyse ist es möglich, die Besuche einer kompletten Webseite auszuwerten ohne Änderungen am Quellcode der Seiten vornehmen zu müssen.

Dadurch, dass der Webserver jeden Seitenaufruf protokolliert lässt diese Methode eine sehr exakte Auswertung einer Webseite und deren Besuche zu. Die aber nicht in Echtzeit erfolgt.

2.2.2 Tags und Pixel

Bei diesem Verfahren werden kleine Grafiken (normalerweise 1² Pixel groß) und/oder Javascript direkt in dem Quellcode der Webseiten eingebettet. Die Grafik kann auf einen, von der überwachten Webseite getrennten, Server ausgelagert werden. In der Regel liegen die Grafikdateien auf einem Webserver des Dienstleisters, der zur Analyse beauftragt wurde.

Mit Hilfe von Javascript ist es möglich, weitere Daten über den Rechner des Besuchers zu erfahren. Unter anderem kann mittels Javascript die Auflösung und Farbtiefe in Erfahrung gebracht werden. Der Javascript-Code sammelt diese zusätzlichen Daten und manipuliert den Aufruf der Grafikdatei dahingehend, dass diese zusätzlichen Daten mit im Query-String übertragen werden. Dieser zusätzliche Javascript-Code wird heute nahezu von jeder Lösung eingesetzt. Ist jedoch unwirksam, wenn der Benutzer die Nutzung von Javascript deaktiviert hat.

Die Einbindung der Grafikdatei in die zu überwachende Webseite könnte folgendermaßen aussehen:

```
<img src="/pfad/zur/ressource/pixel.gif?id=1" />
```

2.2.3 Cookies

Eine Methode um wiederkehrende Besucher einer Webseite zu identifizieren, ist es Cookies auf dem Rechner der Besucher zu speichern. Im einfachsten Fall wird an dem Cookie nur erkannt, ob es sich um einen unbekannten Besucher oder um einen bekannten Besucher handelt. Zusätzlich lassen sich in Cookies noch weitere Daten speichern, die bei jedem Besuch ausgewertet werden können. Ein Browser darf laut den Cookie-Standards das Auslesen eines Cookies nur dem Server erlauben, der es auch gesetzt hat.

Besucher, die innerhalb ihres Browsers Cookies deaktiviert haben, öfter den Browser wechseln, nur für den Besuch zeitlich begrenzte Cookies (Session-Cookies) erlauben oder anfallende Cookies regelmäßig löschen, verfälschen die Auswertung. Im ersten Fall würde jeder Aufruf wie ein neuer Besucher aussehen und die anderen Fälle verhindern eine zweifelsfreie Erkennung von wiederkehrenden Besuchern.

Auch wenn das Setzen von Cookies sehr umstritten ist, existiert kein verbreitetes alternatives Verfahren um wiederkehrende Besucher zu erkennen. Die Cookie-Setzung kann deswegen als Standard bei der Erkennung von wiederkehrenden Besuchern angesehen werden.

2.2.4 Third-Party-Cookies

Ein Cookie, welches nicht von dem Server der aufgerufenen Webseite gesetzt wird, sondern von einem anderen Server, wird Third-Party-Cookie genannt. Cookies, die von dem Server der aufgerufenen Webseite gesetzt werden, werden hingegen als First-Party-Cookies bezeichnet.

Ist in eine Webseite zum Beispiel ein Bild eingebettet welches von einer anderen Domain abgerufen wird und setzt der Server dieser Domain dann einen Cookie, handelt es sich um ein Third-Party-Cookie. Das entscheidende Kriterium ist hier die unterschiedliche Domain, für die ein Cookie gesetzt wird und nicht ob es sich auch physikalisch um unterschiedliche Server handelt.

Normalerweise sollte das Setzen dieser Art von Cookies laut den Cookie-Standards standardmäßig in den Browsereinstellungen deaktiviert sein [2][3]. Keiner der großen Browserhersteller hält sich aber an diese Vorgabe und erlaubt dass setzen dieser Cookies in ihren Standardeinstellungen aber bieten die Möglichkeit zur Deaktivierung durch den Benutzer. [4]

Third-Party-Cookies erlauben das Verfolgen der Benutzeraktivitäten, auf verschiedenen Webseiten, über die Grenzen von Servern und Domains hinaus. Ein großer Dienstleister, wie zum Beispiel Google Analytics, könnte dem Benutzer, den Besuch mehrerer Webseiten nachweisen. Anhand dieser Informationen, hat der Dienstleister ein relativ genaues Bild über das Surfverhalten des Benutzers und kann, unter Umständen, auf einige seiner Interessen schließen. Wertvolle Informationen wenn es darum geht Werbung, die den Besucher ansprechen soll, auf einer Webseite zu platzieren.

Möchte ein Dienstleister auf Third-Party-Cookies verzichten und trotzdem einen Cookie bei dem Benutzer platzieren, kann auf Javascript zurückgegriffen werden. Javascript darf und wird von den Browsern nur akzeptiert, wenn es vom Server der aufgerufenen Seite stammt. Der Dienstleister muss sein Javascript also direkt in der Webseite des Kunden integrieren. Dafür kann er aber mittels dem Javascript First-Party-Cookies setzen und auslesen.

2.2.5 Pro & Contra

Alle drei vorgestellten Verfahren haben ihre Vor und Nachteile. Wobei für die Cookie-Methode keine verbreitete Alternative existiert. Aus Gründen der Übersichtlichkeit liste ich im Folgenden, die einzelnen Vor und Nachteile noch einmal auf.

Webserver-Logdateianalyse

- **+** komplette Protokollierung aller Zugriffe
- **+** Erstellung eines Chronologischen Bewegungsprofil möglich
- **+** Änderungen am Quellcode der Seiten sind nicht nötig
- **+** Identifizierung beliebter und unbeliebter Seiten ist möglich
- **-** keine sofortige Auswertung in Echtzeit
- **-** keine Identifizierung von wiederkehrenden Besuchern

Tags und Pixel

- \+ komplette Protokollierung aller Zugriffe
- \+ Begrenzung auf relevante Bereiche der Webseite
- \+ sofortige Auswertung in Echtzeit
- \+ Identifizierung beliebter und unbeliebter Seiten ist möglich
- \+ Javascript kann zusätzliche Informationen sammeln
- \- Javascript kann vom Besucher deaktiviert werden
- \- Änderungen am Quellcode der Seiten sind erforderlich
- \- keine Identifizierung von wiederkehrenden Besuchern

Cookies

- \+ wiederkehrende Besucher können identifiziert werden
- \+ kann andere Verfahren sinnvoll ergänzen
- \- Cookies können vom Besucher untersagt/gelöscht werden
- \- nur begrenzte Möglichkeiten um Daten zu sammeln

Zu erkennen ist, dass die Möglichkeiten und Grenzen der einzelnen Verfahren durchaus unterschiedlich sind und ein sinnvolles sammeln von Daten mit dem Cookie-Verfahren eigentlich nur in Kombination mit einem der anderen Verfahren möglich ist.

2.2.6 Kombination unterschiedlicher Verfahren

Die serverbasierte Logdateianalyse, wie auch die clientbasierte 'Tags und Pixel'-Lösung, lassen sich mit dem Cookie-Verfahren kombinieren. Eine besonders effektive Variante ist hier die kombinierte 'Tags und Pixel'-Lösung. Sie liefert, bei vergleichbaren Aufwand, die bessere Datenqualität. Dürfen die Benutzerdaten den Server der überwachten Webseite nicht verlassen, wie es bei der 'Tags und Pixel'-Lösung üblicherweise der Fall ist, sollte die kombinierte Logdateianalyse zur Anwendung kommen.

2.3 Methoden

Die Web-Analytics lässt sich in zwei Methoden aufteilen. Zum einen das 'Monitoring,' also das eigentliche sammeln und aufbereiten von Daten. Zum anderen die 'Optimierung', welche sich mit der Optimierung des Webauftrittes beschäftigt.

2.3.1 Monitoring

Monitoring ist das Sammeln und Aufbereiten von Daten, über die Besuche und Besucher der Webseite, um aussagekräftige Informationen zu erhalten. Zum Beispiel kann der Zusammenhang zwischen einer Werbekampagne, zu Gunsten einer Webseite, mit einer veränderten Besucheranzahl in Verbindung gebracht werden.

Ziel des Monitorings ist es aussagekräftige Informationen zu generieren die, richtig interpretiert, Schwachstellen eines Webauftrittes aufzeigen können.

2.3.2 Optimierungen

Bei der Optimierung wird anhand von gesammelten Daten entschieden in wie weit es möglich ist die Seite zu optimieren und in welchen Bereichen angesetzt werden sollte. Über sogenannte Pfad-Analysen können beliebte und unbeliebte Seiten identifiziert und dadurch auf eine eventuell schlechte Benutzerführung innerhalb der Webseite geschlossen werden.

Ziel der Optimierung ist eine effiziente und zielgerichtete Optimierung des Webauftrittes zu erreichen.

2.4 Datenschutz

Aus dem Bereich Datenschutz interessiert mich, in Hinblick auf das zu erstellende System, welche Daten dürfen nach deutschen Recht gespeichert werden und welche nicht. Auch ist es wichtig zu wissen, wo sich Grauzonen befinden, um schon beim Design des Systems mögliche Konflikte mit dem Datenschutz zu vermeiden. Für Interessierte habe ich Links im Quellenverzeichnis hinterlegt, auf denen Informationen und auch Verweise auf Gerichtsurteile usw. zu finden sind. [1][5][6]

Für meine Entscheidungen sind eigentlich nur 2 Punkte relevant. Zum Einen, dass ein Benutzer informiert werden muss, wenn über ihn Daten erhoben werden und das es sich bei der IP-Adresse vielleicht um ein personenbezogenes Datum handelt.

Die Information des Benutzers, über die Erhebung von Daten beim Besuch auf seiner Webseite, liegt in der Zuständigkeit des Webseitenbetreibers. Dieser muss den Besucher über eine Datenschutzerklärung auf seiner Webseite informieren.

Laut dem Telemediengesetz (§12) dürfen keine personenbezogenen Daten über Benutzer, ohne deren Einwilligung, gespeichert werden. Wie den genannten Quellen zu entnehmen ist, ist es nicht endgültig entschieden, ob es sich bei einer IP-Adresse um ein personenbezogenes Datum handelt. Um aber allen möglichen Konflikten aus dem Weg zu gehen, werde ich bei meinem System keine IP-Adressen speichern. Eine Alternative wäre das Maskieren der gespeicherten IP-Adresse, damit keine eindeutige Zuordnung mehr möglich ist. Diesen Schritt hat Google Analytics vor kurzem unternommen. Google bietet den Webseitenbetreibern, die das Angebot nutzen, die Option an IP-Adressen nur noch maskiert zu speichern.

Ein Beispiel wäre das Maskieren des vierten Oktetts einer Ipv4-Adresse.

213.209.191.116 → personenbezogenes Datum
213.209.191.*** → kein personenbezogenes Datum

2.5 Fazit

Ich habe mich für einen Einsatz einer Kombination der 'Tags und Pixel'-Lösung mit Cookies entschieden.

Auf der zu überwachenden Webseite wird ein Bild platziert, welches eine Dimension von einem Pixel² (im nachfolgenden als Pixel bezeichnet) hat. Dieses Pixel befindet sich auf dem Pixel-Server, der zum Erheben der Daten zuständig ist und wird von diesem abgerufen. Dieser Aufruf ist für den Besucher zum größten Teil transparent.

Der Pixel-Server erzeugt für jeden neuen Besucher eine eindeutige Benutzererkennung und speichert diese in einem Cookie auf dem Rechner des Benutzers. Da der Server der Webseite und der Pixel-Server nicht identisch sind, handelt es sich bei dem Cookie um einen sogenannten Third-Party-Cookie. Anhand des gesetzten Cookies ist es möglich, neue Besucher von wiederkehrenden Besuchern zu unterscheiden und damit zurückliegende und neue Besuche einer eindeutigen Benutzerkennung zuzuordnen.

Mit der gewählten Kombination ist gewährleistet, dass die gesammelten Benutzerdaten in Echtzeit ausgewertet werden können. Außerdem können Aufrufe und Daten von den Besuchern über verschiedene Besuche hinweg zugeordnet werden. Besucher werden immer an der eindeutigen generierten Benutzeridentifikation wiedererkannt, weshalb eine Speicherung der IP-Adresse des Benutzers entfällt. Es erfolgt also keine Speicherung von personenbezogenen Daten

Ablauf (Abbildung 2) **:**

1. Der Besucher ruft die Webseite vom Webserver ab. (rote Anforderung)
2. Der Webserver antwortet mit der angeforderten Webseite. (rote Antwort)
3. Das im HTML-Code eingebettete Pixel wird vom Pixel-Server abgerufen. (blaue Anforderung)
4. Der Pixel-Server ermittelt und verarbeitet die Besucherdaten, setzt ein Cookie und antwortet mit der Pixel-Grafik. (blaue Antwort)

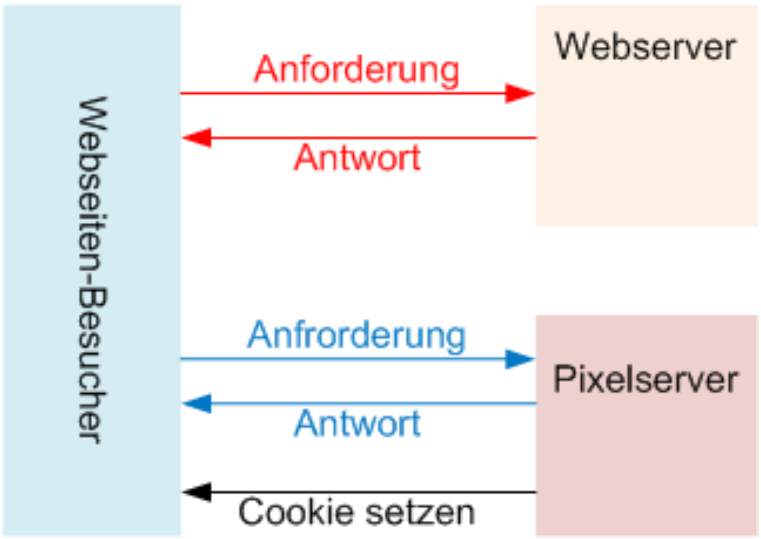

Abbildung 1: Kombination der Verfahren

Verwendete Quellen: [1][2][3][4][5][6][7]

3 Webserver

Es gibt eine Reihe an Webservern, die als 'Pixel-Server' in Frage kommen. Um eine Entscheidung für den richtigen Webserver treffen zu können, habe ich mir vier populäre Webserver genauer angeschaut. Am Ende des Kapitels liste ich noch einmal zusammengefasst alle Vor- und Nachteile der einzelnen Webserver auf und entscheide mich im Fazit für eine Variante.

3.1 Apache HTTP-Server

Der Apache HTTP-Server ist der weltweit meist benutzte Webserver [8]. Er ist ein Modular aufgebauter Webserver, der von dem 'Apache HTTP Server Project' als Open Source weiterentwickelt wird. Für alle gängigen modernen Betriebssysteme wird eine Version angeboten. Zum Produktiveinsatz empfiehlt das Entwicklerteam derzeit die Version 2.2.x [9].

Der Apache HTTP Server kann mit entsprechenden Modulen zum Beispiel als Lastenverteiler für n Tomcat Servletcontainer dienen. Andere Module ermöglichen den HTTP-Server die Fähigkeit der SSL-Verschlüsselung über das HTTPS-Protokoll oder den Einsatz als WebDAV-Server. Weiterhin kann der Apache Webserver dahingehend konfiguriert werden, dass er als Reverse-Proxy dient, um die einzelnen Komponenten des Systems unter einer einheitlichen Domain zusammenzufassen.

Durch die Möglichkeit den Apache HTTP-Server mittels Modulen zu erweitern kann er einen großen Funktionsumfang bieten.

3.2 Apache Tomcat

Der Tomcat Server ist, wie der Apache HTTP Server, ebenfalls Open Source und für die meisten modernen Betriebssysteme erhältlich. Die aktuelle stabile Version, zum jetzigen Zeitpunkt, ist die Version 6.0. Für den Betrieb ist Java der Version 1.5 nötig und der Tomcat unterstützt die Spezifikationen 2.5 der Servlets und 2.1 für JSPs. [10]

Der Apache Tomcat ist ein in Java geschriebener Servletcontainer und bietet eine Umgebung, die es erlaubt Java-Code auf einem Webserver auszuführen. Die Webinhalte werden über Java Servlets und Java Server Pages bereitgestellt. Zusätzlich verfügt der Tomcat über einen kompletten HTTP-Server, der die Inhalte zum Abruf zur Verfügung stellt. Da, im Gegensatz zu seinen Vorgängern, im Tomcat ein Servletcontainer und ein HTTP-Server vereint sind, handelt es sich jetzt um einen vollwertigen Webserver.

Der Tomcat besteht aus zwei Teilen. Zum Einen den Servletcontainer der Catalina bezeichnet wird und dem Connector Coyote, der Anfragen über verschiedene Protokolle entgegennimmt. Um die dynamischen Inhalte des Tomcat auch durch andere Webserver ausliefern zu können, kann der Tomcat über das JServ-Protokoll (AJP) angesprochen werden. Connectoren für dieses Protokoll gibt es für verschiedene Webserver, unter anderem dem Apache HTTP-Server und den Microsoft IIS.

Im Produktionseinsatz wird oft eine Apache HTTP-Server vor den Tomcat geschaltet, der Anfragen auf dynamische Inhalte an den Tomcat über AJP weiterleitet. Gerade bei der Auslieferung von statischen Inhalten hat der Apache HTTP-Server Geschwindigkeitsvorteile gegenüber dem im Tomcat integrierten Coyote. Wie schon erwähnt, kann ein Apache HTTP-Server auch als Lastenverteiler für mehrere Tomcat konfiguriert werden und die HTTPS-Verschlüsselung für den Tomcat übernehmen.

3.3 Microsoft IIS

Die Internet Information Services (aktuelle Version 7.5) von Microsoft bieten mehr als nur einen reinen Webserver. Neben einem HTTP und HTTPS Server ist zum Beispiel noch der Einsatz als FTP, SMTP oder POP Server möglich. Integriert ist eine vollwertige Version der IIS nur in den Server-Betriebssystemen von Microsoft. In den Betriebssystemen für Endanwendern ist, wenn überhaupt, nur eine stark begrenzte Version integriert. [11]

Über IIS ist es möglich ASP und NET Anwendungen zur Verfügung zu stellen. Außerdem ist es möglich einen Tomcat anzubinden oder auch PHP zur Verfügung zustellen. Hierfür werden aber passende Connectoren benötigt.

3.4 Lighttpd Webserver

Der Lighttpd-Webserver ist unter der BSD-Lizenz als freie Software verfügbar. Erhältlich ist er für Unix-Derivate und Windows Betriebssysteme. Zum jetzigen Zeitpunkt ist die Version 1.4 aktuell [12]. Der Lighttpd-Webserver erfreut sich steigender Beliebtheit und hat es mittlerweile geschafft, auf den vierten Platz der meistgenutzten Webserver vorzustoßen [8].

Lighttpd ist ein Webserver, der sich ähnlich wie der Apache HTTP-Server, über Module erweitern lässt. Im Gegensatz zum Apache HTTP-Server, der auf einem Multi-Prozess-Design beruht, kann der Lighttpd-Server mehrere Anfragen parallel in einem Prozess abarbeiten. Dadurch ist er in der Lage mehrere gleichzeitige Anfragen effizienter abzuarbeiten als es der Apache HTTP-Server kann.

Die Anzahl an verfügbaren Modulen ist recht umfangreich. Wie beim Apache HTTP-Server sind auch Module wie zum Beispiel für HTTPS und zur Anbindung eines Tomcat über das JServ-Protokoll verfügbar.

3.5 Pro & Contra

Ich führe in diesem Kapitel nur Vor- und Nachteile auf, die relevant für meine Entscheidungen im Hinblick auf das zu entwickelnde System sind.

Apache HTTP-Server

- \+ frei verfügbar
- \+ für Windows und Unix-Derivate verfügbar
- \+ Einsatz als Reverse-Proxy
- \+ Einsatz als Lastenverteiler
- \+ Anbindung eines Tomcat über JServ
- \- Multi-Prozess-Design

Apache Tomcat

+ frei verfügbar
+ für Windows und Unix-Derivate verfügbar
+ umfangreiche Verarbeitung der Aufrufe durch den Einsatz von Java
+ einsetzbar als vollwertiger HTTP-Server oder als reiner Servletcontainer
- HTTP-Server ist im Vergleich langsamer und nicht so umfangreich
- benötigt sehr viele Ressourcen (Arbeitsspeicher)

Microsoft IIS

+ umfangreiches Angebot an verschiedenen Diensten
+ Anbindung eines Tomcat über JServ
- nicht frei verfügbar
- als Server kommt nur ein Windows-Server in Frage

Lighttpd Webserver

+ frei verfügbar
+ für Windows und Unix-Derivate verfügbar
+ asynchrone Verarbeitung der Anfragen und Verbindungen
- Oftmals wird Lighttpd nur in einer angepassten Version eingesetzt.

3.6 Fazit

Für meine Anwendung habe ich mich für eine Kombination aus einem Apache HTTP-Server und einem Apache Tomcat Servletcontainer entschieden. Der Tomcat wird an den Apache HTTP-Server mittels JServ angebunden.

Gründe:

- Zum einen kann, wie bereits erwähnt, der Apache HTTP-Server als Lastenverteiler dienen, wenn es wegen hohen Lastenaufkommen nötig ist weitere Tomcat einzusetzen.
- Zum anderen möchte ich den Apache HTTP-Server als Reverse-Proxy einsetzen, um alle Bestandteile meiner Anwendung unter einer Domain zusammen fassen zu können. Mein System muss und wird später Javascript einsetzen. Aus Sicherheitsgründen unterbindet Javascript aber den Aufbau von TCP-Verbindungen zu und den Abruf von Scripts von fremden Domains (Cross-Site-Scripting). Diese Einschränkung kann durch den Einsatz als Reverse-Proxy umgangen werden.
- Weiterhin ist es möglich, das Betriebssystem (Unix-Derivate und Windows) der Server frei zu wählen oder zu mischen. Jede Komponente kann auf einem anderen Server mit einem anderen Betriebssystem laufen. Da der Microsoft IIS nur auf einem Windows Betriebssystem zum Einsatz kommen kann, habe ich mich gegen einen Einsatz des ISS entschieden.
- Eine Kombination aus dem Apache HTTP-Server und einem Tomcat Servletcontainer ist in der Praxis häufig eingesetzt und erprobt. Das ist ein Vorteil, da so ein reibungsloses Zusammenspiel der einzelnen Komponenten

gewährleistet ist. Hier liegt auch der Grund, warum ich mich gegen einen Einsatz von Lighttpd entschieden habe.

- Durch den Einsatz eines Tomcat als Pixel-Server, auf dem sich auch das ETL befindet ist eine solide und umfangreiche Verarbeitung der einzelnen Aufrufe gegeben.

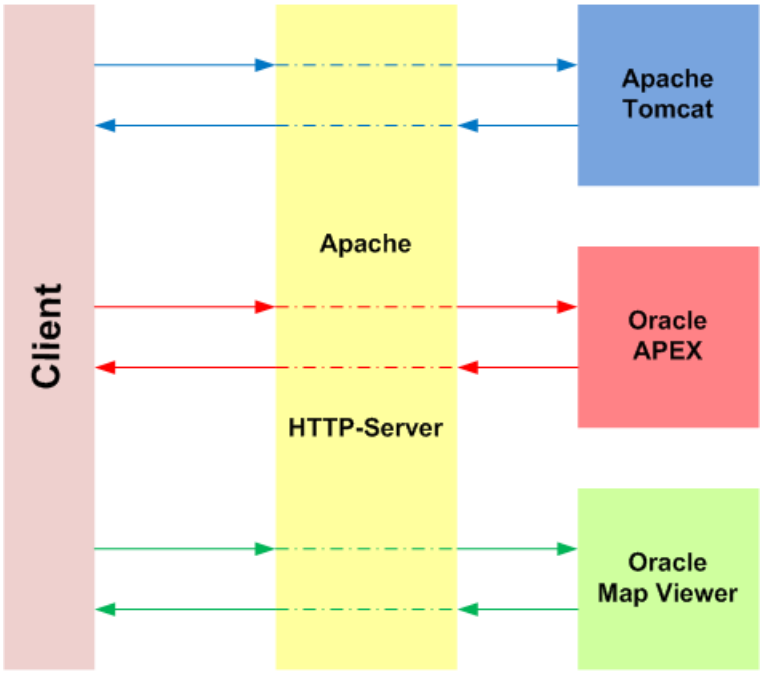

Abbildung 2: Apache HTTP als Reverse Proxy

Verwendete Quellen:[8][9][10][11][12][13][14][15][16]

4 Pixel-Anwendung

Bei der Pixel-Anwendung handelt es sich um eine Webapplikation, die in dem Tomcat Servletcontainer läuft. Die Anwendung ist eine Umgebung, in der Module eingefügt werden können und selbst den Modulen Funktionen zur Verfügung stellt. Die Module sind für die korrekte Auswertung der Aufrufe zuständig. Der Kern der Pixel-Anwendung ist hingegen nur dafür zuständig, eintreffende Requests und neue Session für die Module aufzubereiten.

Ich werde in diesem Kapitel zum größten Teil auf Source-Code verzichten werde aber die wichtigsten Interfaces der Pixel-Anwendung zeigen und genauer auf die Funktion der anhand dieser Interfaces implementierten Klassen eingehen.

4.1 Design der Anwendung

Context-Listener

Der Context-Listener wird aufgerufen, wenn eine Webapplikation gestartet oder gestoppt wird. Für beide Ereignisse implementiert ein Context-Listener über das Interface ServletContextListener geeignete Methoden. Innerhalb der 'web.xml' wird ein Context-Listener dem Servletcontainer bekannt gemacht.

Beim Starten der Pixel-Anwendung werden hier die Schritte ausgeführt, die für die ganze Webapplikation gelten. Es werden Properties gelesen oder auch Daten, die beim letzten Stoppen der Pixel-Anwendung gespeichert wurden, wieder eingelesen.

Beim Stoppen der Pixel-Anwendung werden hier Daten persistent gespeichert, die über mehrere Starts und Stops der Webapplikation gelten sollen.

In meiner Implementation werden vor allem Daten vom Context-Listener gelesen und geschrieben, die eine Übersicht über die Zugriffe auf die Pixel-Anwendung bieten. Die Übersicht kann über die URL 'http://<Domain>/ETL/OverviewServlet' aufgerufen werden.

Session-Listener

Der Session-Listener wird aufgerufen, wenn eine Session innerhalb der Webapplikation erstellt oder zerstört wird. Auch ein Session-Listener implementiert über ein Interface (HttpSessionListener) die hierfür benötigten Methoden. Ein Session-Listener wird ebenfalls über die 'web.xml' Datei dem Servletcontainer bekannt gemacht.

Im Session-Listener befindet sich die Logik, die für alle erstellten Sessions, unabhängig davon, welchem Modul die aufgerufene Ressource gehört, gelten soll. Zum Beispiel wird bei der Erstellung einer Session ein Lock initialisiert und im Session-Scope abgelegt. Dieses Lock synchronisiert bei parallelen Aufrufen, von Ressourcen, die kritischen Bereiche innerhalb der Filter und Servlets. Zusätzlich wird dem Session-Scope noch ein SessionDestroy-Container hinzugefügt. Hier können die einzelnen Module Objekte vom Typ SessionDestroy einfügen, die aufgerufen werden, wenn eine Session zerstört wird. Die SessionDestroy-Objekte, die dem Container hinzugefügt werden sollen, müssen folgendes Interface implementieren:

```
/* Ein Session-Destroy-Objekt übernimmt die Aufagen die ein Modul
 * direkt vor der Zerstörung einer Session ausführen möchte.
 */
public interface SessionDestroy extends Serializable {

      /* Diese Methode wird von der Pixel-Anwendung aufgerufen
       * direkt vor der Zerstörung einer Session.
       */
      boolean destroy();
}
```

Von allen SessionDestroy-Objekten, die sich zum Zeitpunkt der Zerstörung, in diesem Container befinden, wird die destroy-Methode ausgeführt. Diese Mechanik ermöglicht es jedem Modul, auf die Zerstörung einer Session entsprechend zu reagieren.

4.2 Design eines Moduls

Ein Modul für die Pixel-Anwendung besteht aus folgenden Komponenten:

Context-Listener
Hier finden beim Starten und Stoppen der Web-Anwendung Verarbeitungen statt, die nur das Modul betreffen. Zum Beispiel das Anlegen von Modul eigenen Zugriffszählern.

Modul-Filter
In dem Filter finden die eigentlichen Verarbeitungen der Aufrufe statt. Also zum Beispiel die ETL-Aufgaben.

Property-Datei
Eine optionale Datei, die gebraucht wird, wenn zum Start der Webapplikation Daten eingelesen werden sollen.

Ressourcen
Optionale Datei(en), wenn beispielsweise Grafiken das Ziel eines Aufrufes sein sollen.

Damit ein Modul beim Start der Anwendung initialisiert wird und später angesprochen werden kann, muss in der 'web.xml' der Modul-Filter und der Context-Listener dem Servletcontainer bekannt gemacht werden.

Ein Modul ist nur innerhalb der Pixel-Anwendung lauffähig. Es ist darauf angewiesen, dass die Request- und Session-Objekte von der Pixel-Anwendung entsprechend vorbereitet werden.

Ein Beispiel wäre, dass ein Modul eigenen Code ausführen möchte, wenn eine Session zerstört wird, um das Ende der Session in eine Datenbank einzutragen. Für solche Aufgaben wird, wie schon oben beschrieben, bei der Erstellung der Session von der Pixel-Anwendung dem Session-Scope ein SessionDestroyContainer hinzugefügt. Über ein entsprechend implementiertes SessionDestroy-Objekt kann das Modul die Aktualisierung der Datenbank automatisch ausführen lassen, wenn die Session zerstört wird.

Durch den Einsatz von Modulen ist es sehr einfach möglich, verschiedene Verarbeitungen der Aufrufe zu implementieren. Je nach dem welche Daten aus einem Aufruf extrahiert werden sollen, wird ein entsprechendes Modul implementiert.

4.3 Interaktion der Module mit der Anwendung

Ich werde in diesem Abschnitt die Interaktion der Module mit der Anwendung noch einmal verdeutlichen. Die Grafik, weiter unten auf dieser Seite, zeigt die gesamte Pixel-Anwendung (bläulicher Bereich), die zwei Module beinhaltet (orange Bereiche). Die gräulichen Felder, innerhalb der Pixel-Anwendung, sind Funktionen und Objekte, die von der Anwendung zur Verfügung gestellt werden. Der Bereich auf der linken Seite repräsentiert eine unbestimmte Anzahl von Benutzern, die auf die Pixel-Anwendung zugreifen.

- Bei der Initialisierung der Webapplikation wird der Context-Listener der Pixel-Anwendung abgearbeitet und dabei alle relevanten Daten die zum Betrieb der Anwendung benötigt werden generiert und eingelesen.
- Als nächstes werden die Context-Listener für alle Module geladen und auch hier werden alle zum Betrieb der Module nötigen Daten generiert und eingelesen.

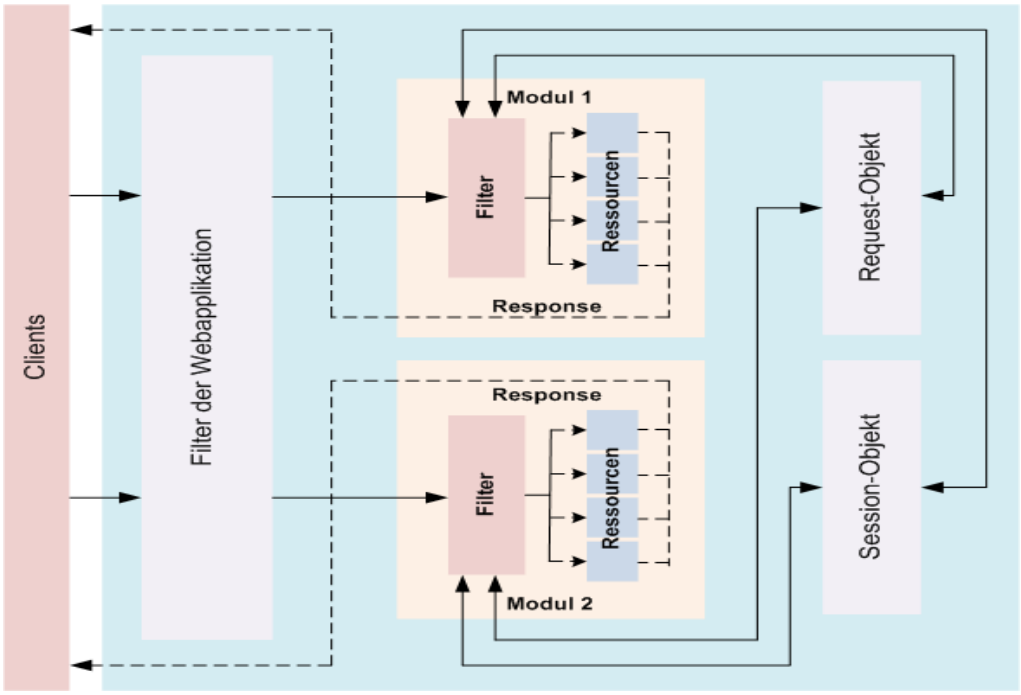

Abbildung 3: Pixel-Anwendung mit zwei Modulen

- Bei der Erstellung einer neuen Session wird das vom Servletcontainer erstellte Session-Objekt von der Pixel-Anwendung für die Benutzung durch die Module vorbereitet.
- Beim Eintreffen eines neuen Requests durchläuft dieser erst die Filter der Pixel-Anwendung und trifft erst danach auf den Filter des Moduls für das der Request gedacht ist. Die eigentliche Verarbeitung des Requests erfolgt im Modul-Filter. Wie Eingangs schon erwähnt, benutze ich die Filter der Anwendung um Statistiken für die gesamte Anwendung, zu erstellen. Diese Übersicht ist besonders für das Debugging wertvoll.
- Jedes Modul greift auf das selbe Session-Objekt zu.

4.4 Extract Transform Load Modul

Um das Extract Transform Load (ETL) für meine Anwendung zu realisieren, habe ich der Pixel-Anwendung ein Modul hinzugefügt, welches die ETL Aufgaben erledigt.

Als Ressource habe ich mich für ein Pixel entschieden. Wird dieses Pixel aufgerufen, trifft der Aufruf zuerst auf einen Filter, in dem die relevanten Daten extrahiert, transformiert und geladen werden. Nach dem Filter wird der Request an ein Servlet weitergeleitet, welches die Pixel-Grafik über den Response ausliefert. Natürlich ist es auch möglich die Pixel-Grafik durch eine andere Ressource auszutauschen.

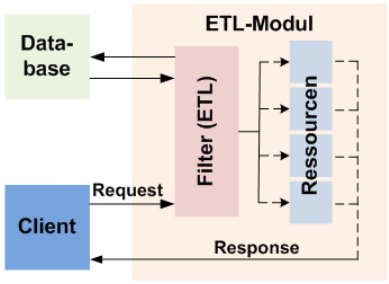

Abbildung 4: ETL-Modul

4.4.1 GeoCity Datenbank

Damit die Standorte der Aufrufer auf einer Karte dargestellt werden können, ist es zwingend erforderlich, Geodaten der Aufrufer zu extrahieren. Ich habe mich bei dieser Aufgabe für die MaxMind GeoLite City Datenbank entschieden um die Daten zu ermitteln.

Die Firma MaxMind [17] bietet verschiedene Datenbanken an, die benutzt werden können, um Informationen über eine IP-Adresse zu erhalten. Neben den kommerziellen Produkten bietet MaxMind auch Open-Source Datenbanken, an die kostenlos zur Verfügung gestellt werden. Für die in dieser Bachelorarbeit, zu lösende Aufgabe kommt die GeoIP City oder die GeoLite City Datenbank in Frage. Wobei die GeoLite City Datenbank die Open Source Variante ist. Außer den Kosten unterscheiden sich die Datenbanken hauptsächlich in der Genauigkeit.

In den folgenden zwei Tabellen habe ich die von MaxMind angegebenen Genauigkeiten für Deutschland und weltweit aufgeführt. [18][19]

Durchschnittliche Genauigkeit weltweit		
Datenbank	**GeoP City**	**GeoLite City**
Genauigkeit mit der IP-Adressen Ländern zugeordnet werden können	99,80%	99,50%
Genauigkeit mit der IP-Adressen, die in einem Radius von 25 Meilen zum tatsächlichen Standort zugeordnet werden können.	83,00%	79,00%

Tabelle 1: GeoCity - durchschnittliche Genauigkeit weltweit

Genauigkeit für Deutschland		
Datenbank	**GeoIPCity**	**GeoLite City**
Prozentsatz der IP-Adressen, die in einen Radius von 25 Meilen zum tatsächlichen Standort zugeordnet werden können.	76,00%	71,00%
Prozentsatz der IP-Adressen, die nicht in einen Radius von 25 Meilen zum tatsächlichen Standort zugeordnet werden können.	22,00%	26,00%
Prozentsatz der IP-Adressen, die zwar Deutschland zugeordnet sind aber keiner Stadt zugeordnet werden können.	2,00%	3,00%

Tabelle 2: GeoCity - Genauigkeit für Deutschland

Die Datenbank wird von MaxMind in zwei Versionen angeboten. Zum Einen als eine Datenbank im binären Format und zum Anderen als eine CSV-Version zum Importieren in eine eigene Datenbank. Das binäre Format wird von MaxMind empfohlen, weil diese aufgrund von speziellen Optimierungen, eine höhere Effizienz erreicht.

Um auf die Datenbank im binären Format zugreifen zu können, bietet MaxMind eine entsprechende API an, die ebenfalls Open Source ist. Die API ist für alle gängigen Programmier- und Script-Sprachen erhältlich.

Ich habe mich für die binäre Datenbank entschieden und diese in dem ETL-Modul der Pixel-Anwendung integriert.

4.4.2 Extract

Beim ersten Aufruf innerhalb einer neu erstellten Session:

Die Erstellung einer neuen Session ist mit einem Lock geschützt um mehrere parallele Aufrufe zu synchronisieren.

1. Der kritische Abschnitt wird über das Lock betreten.
2. Eine neue Session mit einer eindeutigen Sessionidentifikation wird für den Aufrufer erstellt.
3. Die Geodaten des Aufrufers werden aus der GeoLite City Datenbank extrahiert.
4. Es wird geprüft, ob es sich um einen neuen oder um einen wiederkehrenden Besucher handelt.
5. Eine eindeutige Benutzeridentifikation wird generiert bzw. ausgelesen.
6. Dem SessionDestroy-Container der Pixel-Anwendung wird ein SessionDestroy-Objekt hinzugefügt, welches die Aufgaben übernimmt, die beim Zerstören der Session ausgeführt werden sollen. Hierbei handelt es sich um die Aktualisierung des Sessionendes in der Datenbank.
7. Der kritische Bereich wird wieder verlassen.

Bei jedem Aufruf innerhalb einer Session:

Neben der Ressource (Pixel-Grafik) beinhaltet jeder korrekte Aufruf noch eine Webseiten- (sid) und eine Seitenidentifikation (pid). Diese beiden Parameter dienen zur Identifikation der Webseite, in der die aufgerufenen Ressource eingebettet ist.

Webseitenidentifikation	→ Identifiziert eine komplette Webseite.
Seitenidentifikation	→ Identifiziert eine Unterseite einer Webseite.

Ein korrekter Aufruf würde also zum Beispiel folgendermaßen aussehen:

http://<host>:<port>/<path/<resource>?sid=<n>&pid=<n>
http://localhost/ETL/pixel.gif?sid=1&pid=1

Aufrufe, die diese Anforderungen nicht erfüllen, werden direkt beantwortet und nicht ausgewertet.

Handelt es sich um einen korrekten Aufruf, werden die benötigten Daten extrahiert. In meiner Implementation sind es hauptsächlich die Geodaten.

4.4.3 Transform

Beim ersten Aufruf innerhalb einer neu erstellten Session:

Die im Extract-Abschnitt gewonnen oder generierten Daten werden in die benötigten Formate transformiert und innerhalb des geschützten Abschnittes in ein Objekt verpackt, welches eine Session repräsentiert und einem Benutzer/Besucher zugeordnet ist.

Bei jedem Aufruf innerhalb einer Session:

Die im Extract-Abschnitt gewonnen oder generierten Daten werden in die benötigten Formate transformiert und in ein Objekt verpackt, welches einen einzelnen Aufruf (Request) repräsentiert und einer Session zugeordnet ist. Jedem Request wird eine Requestidentifikation zugeordnet, die zumindest innerhalb der übergeordneten Session eindeutig sein muss.

Ein Session-Objekt ist immer einem Besucher/Benutzer zugeordnet und ein Request-Objekt immer einer Session. Damit wird erreicht, dass Aufrufe auch über verschiedene Session hinweg zweifelsfrei einem Benutzer/Besucher zugeordnet werden können. Dieses Vorgehen bietet die Möglichkeit, den Weg, den der Besucher innerhalb einer Webseite genommen hat, nachzuvollziehen. Diese Daten sind unbedingt erforderlich, wenn Klickpfad-Analysen durchgeführt werden sollen.

Die (gekürzten und kommentierten) Interfaces des Session und Request-Objektes:

```
/*
 * Innerhalb eines Session-Objektes befinden sich alle Daten
 * die für die gesamte Session Dauer gelten.
 */
public interface Session extends Serializable {

	/* Gibt die Session-Identifikation zurück, anhand der
	 * eine Session innerhalb der Datenbank identifiziert
	 * werden kann.
	 */
	BigInteger getSessionID();

	/* Gibt die Benutzer-Identifikation zurück, anhand der
	 * ein Benutzer innerhalb der Datenbank identifiziert
	 * werden kann.
	 */
	BigInteger getUserID();

	/* Gibt den Breitengrad des Ortes des Benutzers zurück.
	 */
	float getLatitude();

	/* Gibt den Längengrad des Ortes des Benutzers zurück.
	 */
	float getLongitude();

	/* Gibt den Zeitpunkt zurück, zu dem die Session
	 * erstellt wurde. Also der Zeitpunkt des ersten
	 * Aufrufes einer Ressource innerhalb der Session.
	 */
	Timestamp getStarts();

	/* Gibt den Zeitpunkt zurück, zu dem die Session beendet
	 * wurde.Also der Zeitpunkt des letzten Aufrufes einer
	 * Ressource innerhalb der Session.
	 */
	Timestamp getEnds();

	/* Setzt den Zeitpunkt des letzten Aufrufes einer
	 * Ressource innerhalb einer Session.
	 * date --> Zeitpunkt des letzten Aufrufes.
	 */
	Session setEnds(Timestamp date);

	/* Setzt die Session-Identifikation, anhand der eine
	 * Session innerhalb der Datenbank identifiziert werden
	 * kann.
	 * sessionID --> Session-Identifikation
	 */
	Session setSessionID(BigInteger sessionID);

	/* Setzt die Benutzer-Identifikation, anhand der ein
	 * Benutzer innerhalb der Datenbank identifiziert werden
	 * kann.
	 * userID --> Benutzer-Identifikation
	 */
	Session setUserID(BigInteger userID);
}
```

```
/* Innerhalb eines Request-Objektes befinden sich alle Daten,
 * die für einen einzelnen Aufruf einer Ressource gelten.
 */
public interface Request {

	/* Gibt die Aufruf-Identifikation zurück. Jeder Aufruf
	 * hat seine eigene eindeutige Identifikation.
	 */
	BigInteger getRequestID();

	/* Setzt die Aufruf-Identifikation, anhand der ein
Aufruf
	 * innerhalb der Datenbank identifiziert werden kann.
	 * requestID --> Aufruf-Identifikation
	 */
	Request setRequestID(BigInteger requestID);

	/* Gibt die Identifikation der Session zurück, welcher
	 * der Aufruf zugeordnet ist.
	 * Jede Session hat seine eigene eindeutige
	 * Identifikation.
	 */
	BigInteger getSessionID();

	/* Gibt die Identifikation zurück, mit der eine einzelne
	 * Seite einer kompletten Webseite eindeutig
	 * identifiziert werden kann.
	 */
	int getPageID();

	/* Gib die Identifikation zurück, mit der eine komplette
	 * Webseite eindeutig identifiziert werden kann.
	 */
	int getSiteID();

	/* Gibt den Zeitpunkt zurück, an dem der Aufruf
	 * stattgefunden hat.
	 */
	Timestamp getInvocation();

}
```

Objekte die diese beiden Interfaces implementieren, werden auch im Folgenden Abschnitt benutzt, um die gesammelten und transformierten Daten in die Datenbank zu schreiben.

4.4.4 Load

In dem Load-Abschnitt werden die extrahierten und transformierten Daten in eine Datenbank geschrieben. Bei meinem System ist dies eine Oracle 11g Datenbank. Da die Schreibvorgänge aber über ein Interface erfolgen, ist es auch möglich, Datenbanken anderer Hersteller zu nutzen.

Beim ersten Aufruf innerhalb einer neu erstellten Session:

Wurde bei der Erstellung festgestellt, dass es sich um einen neuen Benutzer handelt, wird ein neuer Benutzer in die Datenbank geschrieben. Nach dem die restlichen Daten extrahiert und transformiert wurden, werden auch die

Session-Daten in die Datenbank geschrieben. Beide Schreibvorgänge werden in dem geschützten Bereich ausgeführt, um die Konsistenz der Daten zu wahren.

Bei jedem Aufruf innerhalb einer Session:

Die extrahierten und transformierten Daten werden einer Session zugeordnet und in die Datenbank geschrieben.

Bei dem zerstören einer Session innerhalb des Servlet-Containers:

Bei jeder Erstellung einer Session wird dem SessionDestroy-Containers ein SessionDestroy Objekt hinzugefügt, welches beim Zerstören der Session den Endzeitpunkt im Session-Objekt setzt und dieses anschließend auch in der Datenbank aktualisiert.

Folgendes Interface (kommentiert und gekürzt) wird für das Schreiben der Daten in die Datenbank genutzt:

```
/* Ein PixelSQL-Objekt übernimmt das Schreiben und Abfragen
 * von Daten in die Datenbank.
 */
public interface PixelSQL {

      /* Prüft, ob die übegebene Benutzer-Identifikation eine
       * gültige Benutzer-Identifikation ist.
       * userID --> Die zu überprüfende Benutzer-Identifik.
       */
      int checkUserID(BigInteger userID);

      /* Jeder gültige Aufruf einer Ressource muss eine
       * Seiten-Identifikation (pid) und eine Webseiten-
       * Identifikation (sid) beinhalten. Anhand dieser
       * Identifikationen wird mit diese Methode der Kunde
       * ermittelt.
       * id       --> Die Seiten (pid) oder Webseiten (sid)ID.
       * isSideID --> Webseiten-Identifikation (true) Seiten-
       *              Identifikation (false)
       */
      int getCoustomerID(int id, boolean isSideID);


      /* Einen neuen Benutzer in die Datenbank eintragen. Die
       * neu generierte Benutzer-Identifikation wird
       * zurückgegeben.
       */
      BigInteger insertUser();

      /* Einen neuen Benutzer in die Datenbank eintragen. Die
       * neu generierte Benutzer-Identifikation wird
       * zurückgegeben.
       * date --> Zeitpunkt an dem der Benutzer zu ersten mal
       *          einen Aufruf getätigt hat.
       */
      BigInteger insertUser(Timestamp date);

      /* Eine neue Session in die Datenbank eintragen.
```

```
         * Anschließend wird die neu generierte Session-
         * Identifikation zurückgegeben.
         * session --> Die Session die neu in die Datenbank   *
                       eingetragen werden soll.
         */
        BigInteger insertSession(Session session);

        /* Eine schon vorhandene Session innerhalb der Datenbank
         * beenden. Die Session wird zum Zeitpunkt des Aufrufes
         * in der Datenbank als beendet gekennzeichnet.
         * Die Rückgabe gibt an, ob das Beenden der Session
         * erfolgreich (true) oder fehlgeschlagen (false) ist.
         * session --> Die Session die in der Datenbank beendet
         * werden soll.
         */
        boolean updateSession(Session session);

        /* Eine schon vorhandene Session innerhalb der Datenbank
         * beenden. Die Session wird zum übergebenen Zeitpunkt
         * in der Datenbank als beendet gekennzeichnet.
         * Die Rückgabe gibt an ob das Beenden der Session
         * erfolgreich (true) oder fehlgeschlagen (false) ist.
         * date     --> Zeitpunkt an der die Session beendet
         *              wurde.
         * session  --> Die Session die in der Datenbank beendet
         *              werden soll.
         */
        boolean updateSession(Session session, Timestamp date);

        /* Eine neue Anfrage anhand des übergebenen Request-
         * Objektes in die Datenbank eintragen.
         * Die neu generierte Aufruf-Identifikation wird
         * zurückgegeben.
         * request --> Die neue Anfrage die in die Datenbank
         *             eingetragen werden soll.
         */
        BigInteger insertRequest(Request request);

        /* Die SQL-Verbindung beenden. Wird die Verbindung
         * erfolgreich beendet, wird true zurückgegeben
         * ansonsten false.
         */
        boolean closeConnection();
    }
```

4.5 Pixel-Servlet

Das Pixel-Servlet ist nur für die Auslieferung der Pixel-Grafik an den Aufrufer über das Response-Objekt zuständig. Nach den ETL-Aufgaben, die in dem Pixel-Filter durchgeführt werden, wird der Request an dieses Servlet weitergeleitet.

Beim ersten Aufruf des Pixel-Servlet, nach dem Start der gesamten Anwendung, wird die Pixel-Grafik mittels einen statischen Initialisierer in eine statische Variable geladen. Das geschieht, weil sich die Daten der Grafik innerhalb des gesamten Zeitraumes, in der die Anwendung läuft, nicht ändern.

Neben dem Laden und Ausgeben der Pixel-Grafik werden hier noch folgende HTTP-Header gesetzt:

Cache-Control: no-cache
Teilt dem Empfänger mit, dass er die Grafikdatei zwar speichern darf aber bei jedem weiteren Aufruf prüfen muss, ob die Datei neu geladen werden muss.

Pragma: no-cache
Sorgt dafür, das Gateways und Proxy-Server das Pixel nicht zwischenspeichern werden. Andernfalls könnte ein Proxy-Server die Anfrage des Pixels direkt beantworten, ohne das die Anfrage an den Pixel-Server weitergeleitet wird und ausgewertet werden kann.

Content-Type: image/gif
Der MIME-Type der Antwort (Response). Teilt dem Client mit, dass die Antwort eine Grafikdatei vom Typ gif ist.

Content-Length: <Größe der Datei in Bytes>
Teilt dem Client mit, wie groß die zu übertragende Grafikdatei ist.

Diese HTTP-Header werden gesetzt, um sicherzustellen, dass der Client weiß, was er bekommt und wie er damit umgehen soll. Werden diese Werte nicht gesetzt, kann es zu verschiedenen unerwünschten Verhalten kommen. Zum Beispiel kann es sein, dass das Pixel gar nicht vom Server abgerufen wird, sondern aus einem Cache (lokal oder extern) kommt oder aber auch ein Browser das Pixel mehrfach abruft (für einen einzelnen Webseiten-Aufruf). Beides führt natürlich zu Verfälschungen bei den gesammelten Daten und bei den späteren Auswertungen.

Je nach Browserhersteller sind die Reaktionen unterschiedlich bei fehlenden Informationen. Besonders beim Mozilla Firefox konnte ich feststellen, dass der Browser auf fehlende Header mit doppelten Aufrufen reagiert.

4.6 Einstellungen: web.xml und context.xml

Die Einträge für die Filter, Listener und Servlets führe ich hier nicht weiter auf. Alle Einträge dieser Art sind normale Standardeinträge und können direkt in der Datei die sich auf der beiliegenden CD befindet eingesehen werden.

Folgenden Eintrag habe ich in der 'web.xml' vorgenommen um die Sessiondauer auf 30 Minuten festzulegen.

```
<session-config>
  <session-timeout>30</session-timeout>
</session-config>
```

Um innerhalb der Anwendung schnelle Datenbankzugriffe ausführen zu können, ohne jedes Mal eine neue Verbindung erstellen zu müssen, habe ich in der 'context.xml' einen Eintrag gemacht, der einen Pool für Datenbankverbindungen erstellt. Es kann sich jederzeit eine offene Verbindung aus diesem Pool entnommen werden, die nach der Benutzung wieder zurückgegeben wird, was sich positiv auf die Performance der

Datenbankzugriffe auswirkt. Der Eintrag beinhaltet alle nötigen Informationen die der Tomcat braucht, um den Pool zu initialisieren und die Verbindungen aufzubauen. Die 'url', der 'user' und das 'password' müssen gegebenenfalls angepasst werden.

```
<Resource
  auth="Container"
  type="oracle.jdbc.pool.OracleDataSource"
  maxActive="20"
  maxIdle="10"
  maxWait="10000"
  name="jdbc/world_sample"
  driverClassName="oracle.jdbc.driver.OracleDriver"
  factory="oracle.jdbc.pool.OracleDataSourceFactory"
  url="jdbc:oracle:thin:@localhost:1521:orcl"
  password="oracle"
  user="world_sample"
  validationQuery="SELECT 1" />
```

4.7 Probleme und Lösungen

Bei meinem Ansatz spielen Cookies eine zentrale Rolle. Zum Einen wird die Sessionidentifikation vom Tomcat in einem Cookie gespeichert und zum Anderen auch innerhalb des Pixel-Filters die Benutzeridentifikation. Dieses System funktioniert nur so lange, wie die Benutzer in ihrem Browser, Cookies nicht deaktiviert haben. Sind die Cookies aber deaktiviert, wird bei jedem Request angenommen, dass es sich um einen neuen Benutzer und eine neue Session handelt. Es wird also bei jedem Aufruf dieses Benutzers eine neue Benutzeridentifikation und Session erstellt und in die Datenbank geschrieben.

Eine Lösung, welches das Problem komplett ausmerzt, gibt es leider nicht. Um die dadurch entstehenden Verfälschungen der Daten abzumildern gibt es folgende Lösungsmöglichkeiten:

- Innerhalb der Auswertung der Daten werden Session und Benutzer mit nur einem Aufruf ignoriert.
- Bei der ETL-Verarbeitung oder bei der Auswertung kann über geeignete Algorithmen versucht werden diese einzelnen Aufrufe anhand der zur Verfügung stehenden Daten einander zuzuordnen.

Verwendete Quellen: [17][18][19][20][21][22]

5 Datenbankdesign

In diesem Kapitel werde ich das Datenbankdesign, anhand eines Datenbankmodelldiagrammes vorstellen. Ich werde jede Tabelle erläutern und später auf die View eingehen, welche die Daten liefert, die im späteren Verlauf, dieser Ausarbeitung, für die Darstellung auf der Karte benötigt werden.

Folgende Anforderungen muss das Datenbankdesign erfüllen:

- Die Benutzer, welche die Webseiten aufrufen müssen gespeichert werden können.
- Der Ort, von dem der Benutzer die Aufrufe gestartet hat, muss gespeichert werden.
- Der Zeitpunkt des Aufrufs und die aufgerufene Seite müssen gespeichert werden.
- Aufrufe müssen einer Seite zugeordnet werden können.
- Eine Seite muss einer Webseite und eine Webseiten einem Kunden zugeordnet sein.

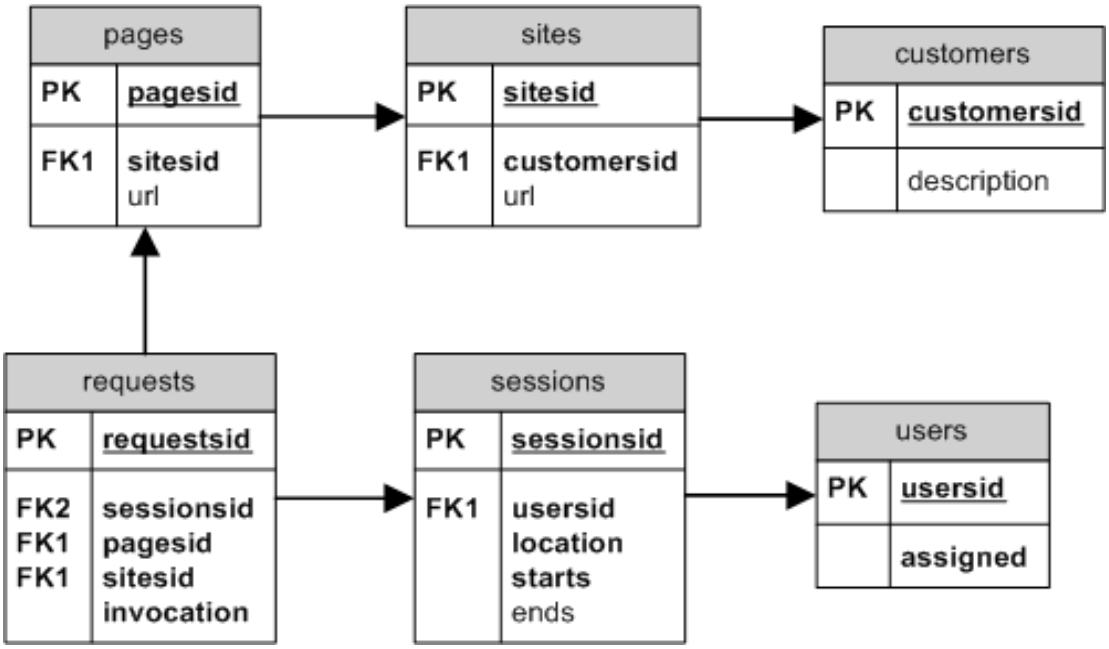

Abbildung 5: Datenbankmodelldiagramm des Datenbankdesigns

customers

In der Tabelle 'customers' werden die Kunden eingetragen. Es kann also n verschiedene Kunden geben. Jedem Kunden wird eine eindeutige Benutzeridentifikation (customersid) zugewiesen.

sites

In der Tabelle 'sites' werden Webseiten den Kunden aus der Tabelle 'customers' zugeordnet. Einem Kunden können somit n verschiedene Webseiten zugeordnet werden. Sinnvoll ist es dem Kunden mindestens eine Webseite zuzuordnen. Jeder Webseite wird eine eindeutige Webseitenidentifikation (sitesid) zugewiesen. Als 'site' gilt eine komplette Webseite mit allen ihren einzelnen Seiten.

pages

In der Tabelle 'pages' werden alle zu überwachenden Seiten einer kompletten Webseite aus der Tabelle 'sites' gespeichert. Somit kann einer kompletten Webseite n verschiedene Seiten zugeordnet werden. Sinnvoll ist es auch hier, jeder Webseite mindestens eine Seite zuzuordnen. Jeder Seite wird eine eindeutige Seitenidentifikation (sitesid) zugewiesen.

users

In der Tabelle 'users' werden alle Benutzer gespeichert die Pixel aufgerufen haben. Die Benutzer werden mit einer eindeutigen Benutzeridentifikation (usersid) gespeichert.

sessions

In der Tabelle 'sessions' werden alle Session gespeichert die ein Benutzer aus der Tabelle 'users' gestartet hat. Jede Session hat einen Startzeitpunkt (starts) und einen Endzeitpunkt (ends). Außerdem werden für jede Session die Geodaten der Position (location) gespeichert, von welchem Ort der Benutzer die Session gestartet hat. Jede Session besitzt eine eindeutige Sessionidentifikation (sessionsid) und jeder Benutzer kann n Session gestartet haben.

requests

In der Tabelle 'requests' werden alle Aufrufe eines Benutzers gespeichert. Jeder Aufruf ist einer Session und einer Seite, die aufgerufen wurde, zugeordnet. Ein Aufruf besitzt einen Zeitpunkt, zu dem der Aufruf ausgeführt wurde. Jedem Aufruf ist eine eindeutige Aufrufidentifikation (requestsid) zugeordnet.

5.1 Views

Um die benötigten Daten aus den Tabellen später in die Auswertung einfließen zu lassen, bieten sich Views an.

Bei der späteren Kartendarstellung möchte ich, zum Beispiel alle offenen Session anzeigen können. Mit anderen Worten alle Benutzer die zu Zeitpunkt des Aufrufes der Karte aktiv auf der zu überwachenden Webseite sind.

Mit dem folgenden SQL-Befehl erzeuge ich eine View, die mir alle relevanten Daten zu den offenen Session liefern kann.

```
CREATE VIEW BA_VIEWS_OPENSESSIONS
  (sessionsid PRIMARY KEY DISABLE NOVALIDATE, usersid,
    assigned, lokation, starts, requests) as
  SELECT
    a.sessionsid,
    a.usersid,
    b.assigned,
    a.location lokation,
    a.starts,
    (SELECT count(*) FROM requests WHERE a.sessionsid =
      sessionsid) requests
  FROM
    sessions a,
    users b
  WHERE
    a.usersid = b.usersid
  AND
    a.ends is NULL
  AND
    a.location is not NULL;
```

Die einzelnen Zellen der View stellen folgende Informationen bereit:

sessionsid
Die Sessionidentifikation der offenen Session

.

usersid
Die Benutzeridentifikation des aktiven Benutzers.

assigned
Das Datum und die Uhrzeit, an dem der aktive Benutzer in der Datenbank registriert wurde.

l**okation**
Die Geodaten des Ortes, von dem aus der Benutzer die Session gestartet hat.

starts
Das Datum und die Uhrzeit, an dem der Benutzer die Session gestartet hat.

requests
Die Anzahl der Aufrufe, die der Benutzer während der aktiven Session durchgeführt hat.

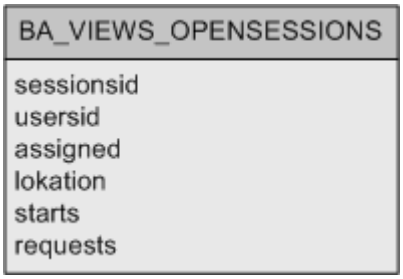

Abbildung 6: offene Session

6 Kartendarstellung

In diesem Kapitel betrachte ich verschiedene Anbieter von Geodaten und/oder Kartenmaterial und deren Darstellung auf einer Webseite. Weiterhin stelle ich den Oracle MapViewer und die WMS-Schnittstelle detailliert vor. Nach der Betrachtung der einzelnen Möglichkeiten ziehe ich mein Fazit und entscheide mich Anhand der Stärken und Schwächen für eine der untersuchten Lösungen.

Am Ende des Kapitels werde ich die gewählte Lösung in eine Webseite des APEX-Webserver integrieren und genauer auf das Zusammenspiel der einzelnen Komponenten eingehen.

6.1 Kartenmaterial und Geodaten

Geodaten beinhalten alle Informationen, um eine Karte zu generieren und räumliche Berechnungen durchzuführen.

Kartengenerierung

Die einzelnen Elemente einer Karte werden über Punkte (Orte), Linien (Wege) und Polygone (Flächen) beschrieben. Eine aus Geodaten generierte Karte besteht also aus einer großen Anzahl an Punkten, Linien und Polygonen, die sich zu einer vollständigen Karte zusammensetzen.

Räumliche Berechnungen

Eine räumliche Berechnung kann zum Beispiel die Berechnung der Distanz zwischen zwei Punkten sein oder aber auch welche 'Points of Interest' sich innerhalb einer festgelegten Distanz zum gewählten Ausgangspunkt befinden. Welche Kunden sich in einem gewählten Umkreis zur Niederlassung befinden, wäre ein Beispiel bei dem die Kunden die 'Points of Interest' wären.

Bei meiner Betrachtung habe ich mich auf drei Anbieter von Geodaten beschränkt. Es gibt aber natürlich noch weitaus mehr Anbieter.

6.1.1 NAVTEQ

Oracle bietet in seinem Technet Geodaten von NAVTEQ zum kostenlosen Download an. Die Geodaten umfassen Ländergrenzen, Bundesländer, Regierungsbezirke, wichtige Fernstraßen und Städte. Diese Daten werden direkt in eine Oracle Datenbank mit der Version von 10g Release 2 oder höher importiert.

Anhand dieser Daten sind umfangreiche räumliche Berechnungen über SQL-Abfragen möglich. Außerdem können mit dem Oracle MapViewer aus diesen Geodaten Karten generiert und nach seinen eigenen Wünschen angepasst werden.

6.1.2 Google Maps

Der bekannteste Anbieter von Kartenmaterial ist sicher Google Maps. Google bietet zwar sein Kartenmaterial über das Internet zum Abruf an aber die dazugehörigen Geodaten sind leider nicht frei verfügbar. Es ist also nicht möglich, das Kartenmaterial seinen eigenen Wünschen anzupassen.

Zur Nutzung des Kartenmaterials ist eine Registrierung erforderlich.

6.1.3 OpenStreetMaps

Das OpenStreetMaps-Projekt bietet ebenfalls umfangreiches Kartenmaterial an. Anders als bei Google, sind auch die kompletten Geodaten die dem Kartenmaterial zugrunde liegen frei verfügbar. Das Kartenmaterial und die Geodaten unterliegen der 'Creative Commons Attribution-Share Alike 2.0'-Lizenz. Diese Lizenz erlaubt jede Art der Nutzung, soweit die Herkunft der Daten angegeben wird und die Software dann selber unter dieser Lizenz steht.

6.2 Darstellung des Kartenmaterials

Um eigene Punkte, also zum Beispiel die Einwahlorte der Webseiten-Benutzer, auf einer generierten Karte darzustellen, werden sogenannte Layer (Abbildung 7) benutzt. Auf diesen Layer werden die entsprechenden Punkte markiert und anschließend über die Karte gelegt. Für eine intuitive Navigation innerhalb der Karte wird in den meisten Fällen eine AJAX-Oberfläche verwendet. Es ist aber möglich, die Karte direkt als Grafik in der Webseite einzubinden. Zum Navigieren muss dann auf Links (links, rechts, hoch, runter) zurückgegriffen werden.

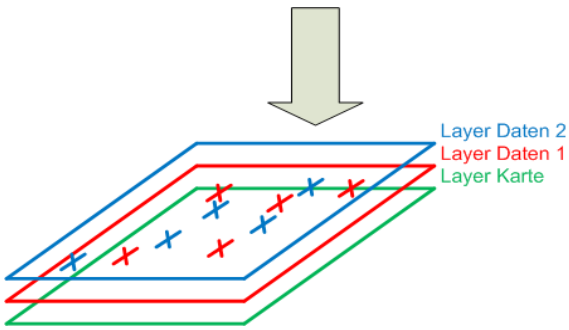

Abbildung 7: Kartensegment und Layer

6.2.1 NAVTEQ

Um aus den NAVTEQ-Geodaten Karten zu generieren und diese später auf der Webseite anzeigen zu können, werden 2 Programme benötigt:

Oracle MapViewer

Der MapViewer wird benötigt, um aus den Geodaten eine Karte zu generieren. Er ist als eine J2EE-Applikation realisiert und muss deswegen nicht auf dem selben Computer wie die Oracle-Datenbank mit den Geodaten laufen.

Oracle MAPS

Um die vom MapViewer generierten Karten auf einer Webseite darstellen zu können, benötigen wir MAPS, welches dem MapViewer beiliegt. MAPS ist eine AJAX-Oberfläche, welche die Darstellung der Karte auf der Webseite übernimmt.

Der MapViewer kann Orte aus einer Oracle Tabelle auslesen und diese auf einen Layer übertragen. Mit Hilfe von MAPS können diese Layer über der Karte eingeblendet oder ausgeblendet werden.

6.2.2 Google Maps

Die Darstellung von Google Maps Kartenmaterial auf einer Webseite erfolgt über eine von Google zur Verfügung gestellte API. Über diese API kann der jeweilige Kartenausschnitt leicht auf der Webseite eingebunden werden. Schwieriger ist es,

eigene Geodaten auf diesem Kartenausschnitt darzustellen. Jeder einzelne Eintrag muss aus der Datenbank ausgelesen werden und dann anschließend mittels Javascript auf der Seite platziert werden. Für kleinere Datenmengen ist dies sicher eine Option, die in Betracht gezogen werden kann, aber bei größeren Datenmengen ist dieses Verfahren einfach zu umständlich.

Um dieses Verfahren zu umgehen, kann uns wieder der MapViewer helfen. Es ist möglich, mit dem MapViewer einen transparenten Layer zu erzeugen, auf dem alle gewünschten Punkte markiert werden. Der erzeugte Layer wird dann anschließend einfach über den von Google Maps gelieferten Kartenausschnitt gelegt. Diese Methode wäre sowohl für kleine aber auch für sehr große Datenmengen geeignet.

6.2.3 OpenStreetMaps

Um Kartenausschnitte, die auf den Geodaten von OpenStreetMaps basieren, auf einer Webseite einzubinden gibt es verschiedene Möglichkeiten.

Einbinden über die Javascript-Bibliothek OpenLayers

Im Prinzip ist dieses ähnlich wie bei der Google Maps API. Über OpenLayers wird der gewünschte und vorberechnete Kartenausschnitt vom OpenStreetMaps Server abgerufen und über eine AJAX-Oberfläche auf der Webseite eingebunden. Auch hier müssen alle gewünschten Geodaten einzeln dem Kartenausschnitt mittels Javascript hinzugefügt werden.

Download des gesamten Kartenmaterials von OpenStreetMaps

Das Kartenmaterial liegt in Form von Geodaten vor, aus denen von geeigneten Programmen Karten erzeugt werden können. Leider gibt es für diese Rohdaten keine Version, die sich direkt in eine Oracle-Datenbank importieren ließe.

Web Map Service (WMS)

Eine Anfrage an einen WMS-Server liefert den gewünschten Kartenausschnitt als Antwort zurück, der dann angezeigt werden kann. Unter anderem bietet der Oracle MapViewer die Möglichkeit Kartenmaterial über WMS abzufragen. Die Darstellung auf der Webseite übernimmt bei dieser Möglichkeit wieder MAPS.

6.3 Web Map Service

Web Map Service (WMS) ist ein Schnittstelle, über die ein Client mit einem Kartenserver kommunizieren kann. Zur Kommunikation zwischen dem Client und dem Server wird das HTTP-Protokoll eingesetzt.

Laut den Spezifikationen kann ein Client drei verschiedene Anfragen an den Server senden.

GetCapabilities

Liefert Metainformationen zurück, die Angaben zum Anbieter, dem Umfang des Angebotes und die möglichen Ausgabeformate enthält. Die Antwort ist im XML-Format verfasst.

GetMap

Liefert das angeforderte Kartensegment im gewünschten Format.

In der Anfrage wird das gewünschte Ausgabeformat, das Koordinatensystem und das gewünschte Kartensegment spezifiziert.

GetFeatureInfo

Mit dieser optionalen Anfrage können thematische Daten abgerufen werden.

Die der Karte zugrunde liegenden Geodaten, werden also direkt auf dem WMS-Server interpretiert und aus diesen, das gewünschte Kartensegment im gewünschten Format erzeugt und an den Client übertragen. Auch bei dieser Variante hat der Benutzer geringen Einfluss auf die Generierung und das Aussehen des Kartenmaterials.

6.4 Vorteile und Nachteile

Google Maps und OpenLayers

- \- Festlegung auf Kartenmaterial eines Anbieters
- \- umständliches einfügen von eigenen Geodaten
- \- eine Registrierung ist erforderlich
- \+ AJAX-Oberfläche
- \+ einfache Benutzung und Einrichtung
- \+ immer aktuelles Kartenmaterial

Kartenmaterial auf Basis von Rohdaten (NAVTEQ/OpenStreetMaps)

- \- Festlegung auf Kartenmaterial eines Anbieters
- \- Software zum rendern der Kartenausschnitte nötig
- \- Daten müssen auf einem aktuellen Stand gehalten werden
- \- höherer Aufwand bei der Einrichtung
- \+ einfaches Einfügen von größeren Mengen eigener Geodaten
- \+ eigene Wünsche bei dem Aussehen der Karten sind möglich

Vorgefertigtes Kartenmaterial über WMS

- \- Software für das Abrufen und die Darstellung nötig
- \- nur begrenzte Möglichkeiten um Einfluss auf die Darstellung zu nehmen
- \+ immer aktuelles Kartenmaterial
- \+ je nach Wahl der Software ist einfaches einfügen von größeren Mengen eigener Geodaten möglich.

6.5 Oracle MapViewer und Oracle MAPS

Der MapViewer ist in der Verbindung mit MAPS eine sehr vielseitige Kombination. Es kann von einem WMS-Kartenserver Kartenmaterial bezogen werden. Aber auch das Erstellen von Kartenmaterial anhand von Geodaten aus einer Oracle Datenbank ist möglich. Ein Wechsel vom Anbieter des Kartenmaterials ist dadurch mit wenig Aufwand zu bewerkstelligen. Empfangene und generierte Kartensegmente werden, wenn gewünscht, in einem Cache abgelegt, um erneute Anfragen für das selbe Segment schneller bearbeiten zu können.

Neben dem Abrufen/Erstellen und Ausliefern der Karten können mit dem MapViewer auch Layer erzeugt werden, die über die eigentliche Karte gelegt werden (Abbildung 7). Die benötigten einzelnen Geodaten, der zu markierenden Orte, werden aus einer vorher definierten Tabelle oder Ansicht einer Oracle-Datenbank entnommen.

Der MapViewer ist als Java-Applikation realisiert und wird unabhängig von der Oracle Datenbank betrieben. Auf dem MapViewer Server wird vom Client über eine TCP/IP-Verbindung zugegriffen und zur Konfiguration eine Weboberfläche angeboten. Mit der Datenbank, in der sich die Tabellen mit den eigenen Geodaten befindet, wird auch über TCP/IP-Verbindung kommuniziert. Es ist also nicht erforderlich, dass sich der MapViewer auf dem selben Server befindet wie der APEX-Webserver oder die Datenbank.

Die eigentliche Kartendarstellung auf der Seite des Client übernimmt das Programm Oracle MAPS welches mit dem MapViewer mitgeliefert wird. MAPS ist eine Javascript-Bibliothek, die Kartensegmente vom MapViewer abruft und in einer AJAX-Oberfläche darstellt. Über diese Oberfläche ist es intuitiv möglich, durch die Karte zu navigieren. Weiterhin kann MAPS Layer vom MapViewer anfordern, über das Kartensegment legen, ein- ausblenden, und aktualisieren.

6.6 Fazit

Alle hier angesprochenen Möglichkeiten lassen sich mit APEX umsetzen. Jede Variante hat ihre Vorteile und Nachteile, die ich im vorigen Abschnitt dargestellt habe.

Folgende Eingenschaften sind für eine Lösung wünschenswert:

- Auswahl bei dem Kartenmaterial
- Auswahl bei dem Anbieter des Kartenmaterials
- wenig eigener Aufwand um das Kartenmaterial aktuell zu halten
- AJAX-Oberfläche für die Darstellung auf der Webseite
- Einfügen von größeren Mengen eigener Geodaten aus der Datenbank
- einfache Aktualisierung der eingebundenen Karte auf der Webseite

Die größte Trefferquote mit der Liste, wird durch eine Kombination der verschiedenen Lösungen erreicht. Mit der Software MapViewer in Zusammenspiel mit MAPS ist es möglich, jedes Kartenmaterial der vorgestellten Lösungen zu nutzen und anzuzeigen. Da wir nicht auf Kartenmaterial angewiesen sind, welches lokal gespeichert ist, haben wir die Möglichkeit ein vorgefertigtes aktuelles Kartenmaterial zu nutzen, ohne den Aufwand einer eigenen Pflege. Außerdem sollte in den meisten Fällen der Aufwand, um den Anbieter des Kartenmaterials zu wechseln, sehr gering sein.

Die MAPS Javascript-Bibliothek bietet eine einfach zu bedienende AJAX-Oberfläche und eine einfache Möglichkeit zum ein/ausblenden und aktualisieren der eigenen Geodaten. Der MapViewer erzeugt für Geodaten aus einer Oracle-Tabelle Layer, die über den Kartenausschnitt gelegt werden und alle Geodaten aus der Datenbank enthalten. Diese Layer werden vom MapViewer erzeugt und im endgültigen Zustand von MAPS an den Client geliefert. Ein umständliches einfügen der einzelnen Geodaten über Javascript entfällt somit.

Meine Wahl fällt aus den genannten Gründen auf den MapViewer in Kombination mit MAPS. Bei dem Kartenmaterial lege ich mich für die Ausarbeitung auf einen Web Map Service mit OpenStreetMap-Geodaten fest, um auch den Vorteil eines aktuellen Kartenmaterials ohne Pflegeaufwand zu haben. Als WMS-Anbieter habe ich den kostenfreien Service 'OMS WMS basic' [23] der WhereGroup [24] gewählt. Neben dem kostenfreien Service, bietet die WhereGroup auch kostenpflichtige Dienste an, die dann zusätzliche Features wie eine garantierte Verfügbarkeit oder eine komplett freie Konfigurierbarkeit bieten.

Auf das Zusammenspiel vom Oracle MapViewer, Oracle MAPS und WMS werde ich im folgenden Kapitel noch genauer eingehen.

6.7 Zusammenspiel der Komponenten

Auf der Abbildung (Abbildung 8) ist der Aufruf der APEX-Anwendung und aller dazugehörigen Aufrufe beschrieben.

1. Der Browser ruft die Webseite vom APEX-Webserver auf.
2. Die Webseite wird von APEX-Webserver an den Browser ausgeliefert.
3. MAPS wird vom Browser gestartet.
4. MAPS ruft den gewünschten Kartenausschnitt vom MapViewer ab.
5. Der MapViewer fordert den Kartenausschnitt vom WMS-Server an.
6. Der WMS-Server liefert den Kartenausschnitt an den MapViewer aus.
7. Der MapViewer leitet den Kartenausschnitt an MAPS weiter.
8. MAPS fragt einen Layer mit eigenen Geodaten beim MapViewer an.

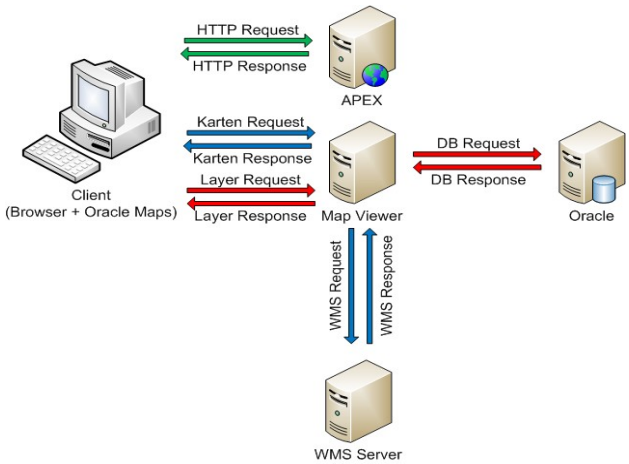

Abbildung 8: Zusammenspiel der Komponenten

9. Der MapViewer stellt eine Anfrage nach den Geodaten an die Datenbank.
10. Die Datenbank beantwortet die Anfrage vom MapViewer mit den Geodaten.
11. Der MapViewer konstruiert aus den Daten einen Layer und liefert ihn MAPS.
12. MAPS zeigt Karte und Layer beim Client in einer AJAX-Oberfläche an.

Verwendete Quellen:[23][24][25][26][27][28][29]

7 Oracle Application Express

Oracle Application Express (APEX) ist eine Umgebung, die direkt in einer Oracle-Datenbank integriert ist. Zum Erstellen einer Webseite, sind keine tieferen Programmierkenntnisse nötig. Die komplette Erstellung der Webseite erfolgt über ein Webinterface, dass APEX zur Verfügung stellt. Die Webseiten werden in der Oracle-Datenbank gespeichert und beim Abruf dynamisch innerhalb der Datenbank aufgebaut und ausgeliefert.

Für mein System setze ich Oracle 11g ein. 11g wird standardmäßig mit APEX Version 3.2.1 ausgeliefert. Es ist aber auch möglich APEX separat herunterzuladen und mit jeder anderen Oracle-Datenbank zu betreiben. Zu beachten ist welche Datenbank-Version mit welcher APEX-Version kompatibel ist. Zu diesem Thema gibt die Oracle-Webseite Aufschluss. Oracle APEX ist eine Weiterentwicklung der Oracle HTML DB und ist mit der Version 2.2 gestartet. Ein Webserver (PL/SQL-Server), der die Auslieferung der generierten APEX-Webseiten übernimmt, wird mit 11g ebenfalls mitgeliefert.

Zu dem Zeitpunkt, an dem ich meine Bachelorarbeit begonnen habe, war die Version 3.2.1 die aktuelle Version, welche mittlerweile durch die Version 4.0 abgelöst wurde. Aus diesem Grund setze ich bei meiner Arbeit die Version 3.2.1 ein.

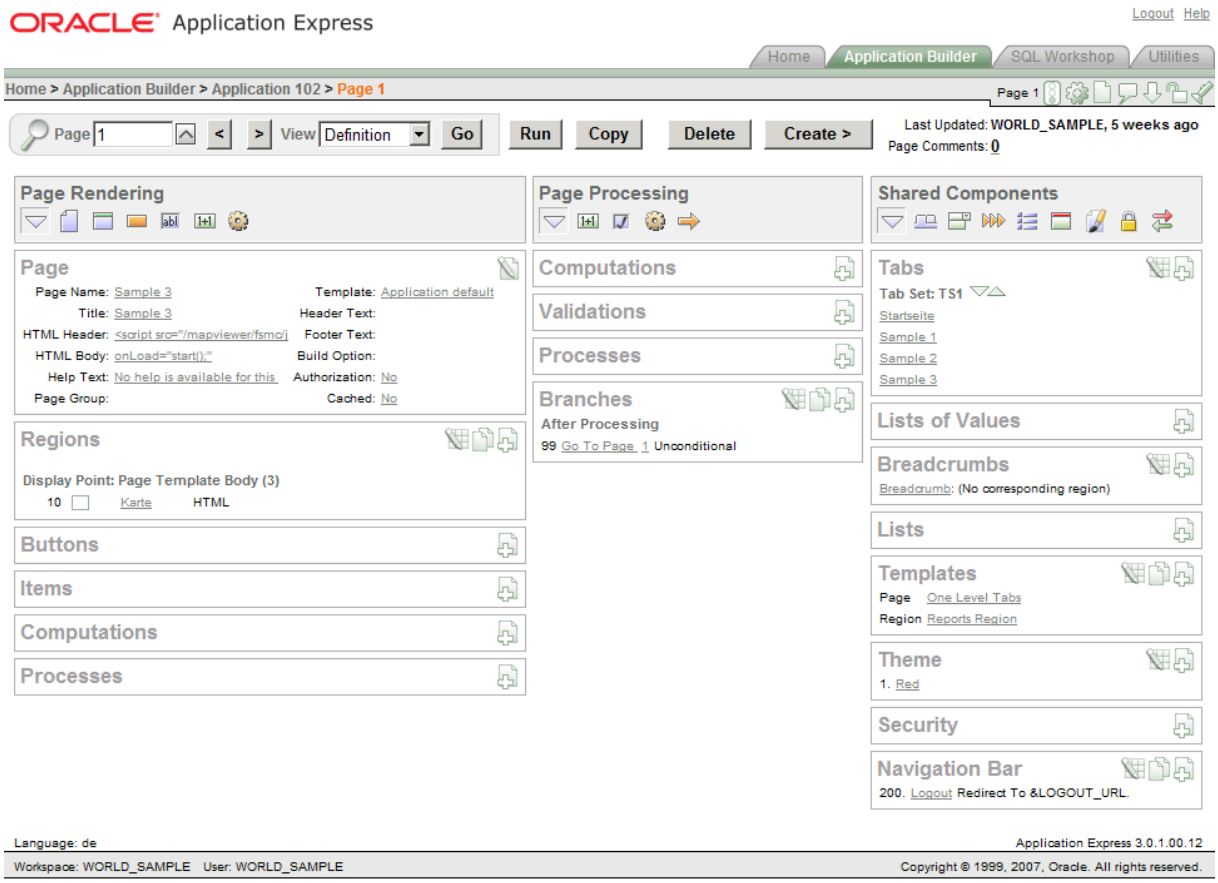

Abbildung 9: Erstellen einer Seite mit dem Oracle Application Express

Mit APEX ist es sehr einfach auf eine Datenbank zuzugreifen und mit wenigen Handgriffen Reports und Statistiken zu erstellen. Selbst die SQL-Programmierkenntnisse, die in Version 3.2 noch für die Abfragen der Reports und Statistiken nötig sind, entfallen mit Version 4.0. Diese Eigenschaften machen APEX natürlich für die Web-Analytics äußerst interessant. Mit wenig Zeitaufwand lassen sich aussagekräftige Reports für die gesammelten Daten erstellen

und präsentieren. Sogar die Kunden selbst könnten sich Reports und Statistiken nach eigenen Wünschen und Bedürfnissen erstellen, ohne fundierte Programmier- bzw. SQL-Kenntnisse haben zu müssen.

Natürlich ist es mit APEX auch möglich eine herkömmliche Webpräsenz zu erstellen, die dynamisch generiert wird und Schreib und Lesevorgänge auf die Datenbank zulässt. Die Oracle-Webseite ist hier ein äußerst beeindruckendes Beispiel, wozu APEX in der Lage ist.

Verwendete Quellen: [30]

8 Komplettes System

In diesem Kapitel behandle ich die Konfiguration der einzelnen Komponenten und deren Zusammenführung. Bei der Konfiguration gehe ich davon aus, dass alle Komponenten auf einen Windows-Server laufen. Im Normalfall unterscheiden sich die Konfigurationen zu einem Unix-Derivat nur durch Pfadangaben und Bibliotheksnamen. Die Entwicklung des Systems habe ich mit einem Windows Server durchgeführt und das Demonstrationssystem auf einem Linux Debian Server installiert.

8.1 Konfiguration: Oracle 11g

Eine weitere Konfiguration neben der Standardinstallation ist nicht notwendig. In meiner Ausarbeitung gehe ich aber von folgenden Einstellungen aus:

Port → 1521
SYSTEM-Passwort → oracle

8.2 Konfiguration: Oracle APEX

Oracle 11g wird standardmäßig mit der APEX Version 3.2.1 ausgeliefert. Aktiviert ist APEX aber nicht automatisch, sondern muss vom Benutzer selber aktiviert werden. Die Konfiguration und Aktivierung ist aber denkbar einfach. Als erstes muss das mitgelieferte SQL-Script 'apxconf.sql', welches sich im '$ORACLE_HOME/apex' Verzeichnis befindet, ausgeführt werden. Während der Ausführung, wird nach dem gewünschten HTTP-Port und Admin-Passwort gefragt (im weiteren gehe ich von Port 8080 und dem Passwort 'oracle' aus). Wurde das Script erfolgreich abgearbeitet, muss nur noch der ANONYMOUS-Benutzer freigeschaltet werden. Das wird mit folgenden SQL-Befehl bewerkstelligt. Wird dieser Schritt nicht durchgeführt, kann APEX nicht angesprochen werden.

```
ALTER USER ANONYMOUS ACCOUNT UNLOCK
```

Sind die beiden Schritte erfolgreich ausgeführt ist APEX unter
http://localhost:8080/apex/
erreichbar.

Das Admin-Interface von APEX ist unter
http://localhost:8080/apex/apex_admin/
erreichbar.

Das Admin-Interface wird benötigt, um einen Workspace anzulegen. Ist dies geschehen, kann unter der erstgenannten URL in den Workspace eingeloggt und eine Webseite angelegt werden, die wir später für die Kartendarstellung benötigen.

Eine weitere Konfiguration ist nicht mehr erforderlich.

8.3 Konfiguration: Oracle MapViewer

Der Oracle MapViewer ist eine eigenständige J2EE-Anwendung. Deshalb müssen wir sicherstellen, dass auf dem Server, auf dem der MapViewer gestartet werden soll, mindestens Java SDK 1.5 installiert ist. Um den MapViewer zu starten, liegt der Installation eine 'start.bat' (Windows) und eine 'start.sh' (Unix) Datei bei. Wenn die

Java-Installation über eine Systemvariable bekannt gemacht wurde, kann der Server ohne weitere Schritte direkt über diese Dateien gestartet werden. Ansonsten müssen die Dateien mit dem Pfad zur Java-Installation angepasst werden.

Nach erfolgreichen Start des MapViewer ist dieser unter
http://localhost:8888/mapviewer
erreichbar.

Um die Geodaten der Besucher auf der Karte darstellen zu können, muss für den MapViewer ein Theme in der Oracle Datenbank erstellt werden. Dieses Theme kann dann von Oracle MAPS aufgerufen und als Layer über die Karte gelegt werden. Das Theme kann von Hand mit dem Oracle Map Builder erstellt oder über ein normalen SQL-INSERT eingefügt werden.

```
INSERT INTO USER_SDO_THEMES (
 NAME,            // Name des Theme
 DESCRIPTION,     // Beschreibung des Theme
 BASE_TABLE,      // Name der Tabelle/View mit den Daten
 GEOMETRY_COLUMN, // Name der Spalte mit den Geodaten
 STYLING_RULES    // Regeln zur Darstellung der Markierungen
) VALUES (
 'OPENSESSIONS',
 null,
 'BA_VIEWS_OPENSESSIONS',
 'LOKATION',
 '<?xml version="1.0" standalone="yes"?>
  <styling_rules caching="NONE" highlight_style="MDSYS:C.BLUE">
   <hidden_info>
    <field column="USERSID" name="UserID"/>
    <field column="ASSIGNED" name="- Erstellt"/>
    <field column="SESSIONSID" name="SessionID"/>
    <field column="STARTS" name="- Erstellt"/>
    <field column="REQUESTS" name="- Aufrufe"/>
   </hidden_info>
   <rule>
    <features style="M.CYAN PIN"> </features>
   </rule>
  </styling_rules>'
);
```

Zusätzlich muss den MapViewer noch Kartenmaterial zur Verfügung gestellt werden. Im Kapitel 'Kartendarstellung' habe ich mich für Kartenmaterial von OpenStreetMap entschieden, welches ich über einen WMS-Service einbinden möchte. Bevor externes Kartenmaterial eingebunden werden kann, muss eine 'Data Source' angelegt werden. In dieser Data Source werden Metadaten zum WMS-Dienst gespeichert. Das Anlegen einer Data Source kann über das Webinterface aber auch über die Datei 'mapViewerConfig.xml', welche sich im Unterverzeichnis 'config' der MapViewer Installation befindet, erfolgen. Wobei der Weg über die Konfigurationsdatei vorzuziehen ist. Über das Webinterface angelegte Data Source werden nicht persistent gespeichert und müssen nach jedem Server-Neustart neu angelegt werden.

Wenn sich der MapViewer auf dem gleichen Server befindet wie die Oracle-Datenbank und alle Komponenten so wie in den vorangegangenen Abschnitten konfiguriert werden, kann eine persistente Data Source mit folgenden Eintrag in der 'mapViewerConfig.xml' erzeugt werden.

```
<map_data_source name="world_sample"
                 jdbc_host="localhost"
                 jdbc_sid="orcl"
                 jdbc_port="1521"
                 jdbc_user="world_sample"
                 jdbc_password="!oracle"
                 jdbc_mode="thin"
                 number_of_mappers="3"
                 allow_jdbc_theme_based_foi="true"
/>
```

Weiterhin muss die Option 'save_images_at' in der 'mapViewerConfig.xml' angepasst werden. Ansonsten können keine Grafiken für Markierungen von Oracle MAPS abgerufen werden, wenn der MapViewer über einen Reverse-Proxy angesprochen wird und der Port 8888 nicht freigegeben ist.

```
<save_images_at file_prefix=""
  url="http://sub.domain.de/mapviewer/images"
  path="/home/oracle/mapviewer/web/images" />
```

Der Pfad und die URL müssen entsprechend angepasst werden.

Um das Kartenmaterial einzubinden, sind folgende Einstellungen im Oracle MapViewer Webinterface vorzunehmen:

1. Im MapViewer Webinterface muss unter dem Menüpunkt 'Manage Map Tile Layers' eine neue Quelle eingetragen werden. Bei einer internen Quelle wird auf Geodaten in einer Datenbank zurückgegriffen, aus denen der MapViewer Kartenmaterial erzeugt. Über externe Quellen wird fertiges Kartenmaterial eingebunden. Da ich mich für Kartenmaterial von einem WMS-Server entschieden habe, ist 'external' die richtige Wahl.
2. Im folgenden Formular werden alle Einstellungen vorgenommen, die der MapViewer braucht, um das Kartenmaterial über den WMS-Service zu beziehen. Da sich Kartenmaterial innerhalb kurzer Zeitspannen nicht drastisch ändert, sollte zu Gunsten der Performance, der Zeitraum für den Kartensegment-Cache nicht zu kurz gewählt werden. Ich habe hier 168 Stunden, also eine Woche, gewählt.

Eine ausführliche Anleitung wird in einem Tutorial von Oracle beschrieben.[25]

Nach diesen Schritten ist der MapViewer jetzt konfiguriert und einsatzbereit. Oracle MAPS kann nun die erforderlichen Kartensegmente und Markierungen anfordern.

8.4 Konfiguration: Apache Tomcat

Da der Apache Tomcat nur über den Apache HTTP-Server erreichbar sein soll, und nicht direkt aus dem Internet angesprochen wird, ist es ratsam den Connector für das HTTP-Protokoll zu deaktivieren. In der Standardkonfiguration (server.xml) sind zwei Connectoren aktiviert. Auf Port 8080 ist der HTTP-Connector konfiguriert und auf Port 8009 befindet sich der AJP-Connector.

Zum deaktivieren wird folgende Zeile der 'server.xml' gelöscht oder auskommentiert:

```
<Connector port="8080" protocol="HTTP/1.1"
 connectionTimeout="20000" redirectPort="8443" />
```

Der AJP-Connector wird für die Anbindung des Tomcat an den Apache HTTP-Server gebraucht. Aus diesem Grund bleibt die folgende Zeile unverändert:

```
<Connector port="8009" protocol="AJP/1.3"
 redirectPort="8443" />
```

Weitere Konfigurationen sind beim Apache Tomcat nicht mehr erforderlich.

8.5 Konfiguration: Apache HTTP-Server

Damit die einzelnen Komponenten zusammenarbeiten, werden alle Komponenten, die über HTTP erreicht werden sollen, auf einer Domain zusammengefasst. Der Apache HTTP-Server muss deswegen als Reverse-Proxy (MapViewer und APEX) und für JServ (Tomcat) konfiguriert werden. Ich gehe von einer Standardkonfiguration aus, bei der virtual Hosts aktiviert sind.

Um den Tomcat über JServ anzubinden, wird als erstes eine Datei workers.properties erstellt, in der alle verfügbaren Tomcat Server aufgeführt und konfiguriert werden. Bei meiner Implementierung des Systems, ist nur eine Tomcat-Instanz vorhanden. Sollten weitere, wegen erhöhten Lastaufkommen, benötigt werden, müssen diese zusätzlichen Instanzen auch in dieser Datei konfiguriert werden.

```
# Jedem Tomcat wird ein Name zugeordnet
worker.list=tomcat1

# Der Port unter dem der Server erreichbar ist
worker.tomcat1.port=8009

# Der Host unter dem der Server erreichbar ist
worker.tomcat1.host=localhost

# Die Kommunikation mit dem Tomcat läuft über das AJP-
# Protokoll der Version 1.3
worker.tomcat1.type=ajp13
```

Weiterhin müssen zur JServ-Anbindung noch folgende Einträge in der 'httpd.conf' des Apache HTTP-Servers vorgenommen werden.

```
# Laden des erforderlichen Moduls zur Kommunikation
LoadModule jk_module "C:/Apache2/modules/mod_jk.so"

# Angabe des Pfades der erstellten workers.properties
JkWorkersFile "C:/Apache2/conf/extra/workers.properties"

# Angabe wo das Log gespeichert werden soll
JkLogFile "C:/Apache2/logs/mod_jk.log"

# Ab welchen Level soll ein Log-Eintrag erstellt werden.
JkLogLevel error
```

Damit der Apache HTTP-Server als Reverse-Proxy fungieren kann, müssen folgende Module in der 'httpd.conf' geladen werden:

```
# Laden der erforderlichen Moduls zur Konfiguration als
# Reverse-Proxy
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_html_module modules/mod_proxy_html.so
LoadModule headers_module modules/mod_headers.so
LoadFile "C:/Server/Apache2/bin/libxml2.dll"
LoadModule xml2enc_module modules/mod_xml2enc.so
```

Zum Schluss müssen noch alle Einstellungen vorgenommen werden, die dem Apache HTTP-Server mitteilen, welche Aufrufe er an welche Komponenten weitergeleitet soll. In diesem Abschnitt werden die Weiterleitungsregeln für den MapViewer, APEX und dem Tomcat konfiguriert.

```
<VirtualHost <IP-Adresse>:<Port>>

  # Pfad an dem zum Beispiel zusätzliche statische Ressourcen
  # liegen
  DocumentRoot /Server/htdocs/domain/

  # Von welchen Domainnamen Aufrufe angenommen werden sollen
  ServerName domain.de

  # Proxy-Zugriffe erlauben
  ProxyRequests off
  <Proxy *>
    Order deny,allow
    Allow from all
  </Proxy>

  # http://domain.de/apex wird an http://domain.de:8080/apex
  # weitergeleitet.
  ProxyPass /apex http://domain.de:8080/apex
  ProxyPassReverse /apex http://domain.de:8080/apex

  # http://domain.de/i wird an http://domain.de:8080/i
  # weitergeleitet.
  ProxyPass /i http://domain.de:8080/i
  ProxyPassReverse /i http://domain.de:8080/i

  # http://domain.de/mapviewer wird an
  # http://domain.de:8888/mapviewer weitergeleitet.
  ProxyPass /mapviewer http://domain.de:8888/mapviewer
  ProxyPassReverse /mapviewer http://domain.de:8888

  # Alle Anfragen die unter http://domain.de/ETL ankommen
  # werden an den Tomcat weitergeleitet
  JkMount /ETL tomcat1
  JkMount /ETL/* tomcat1

</VirtualHost>
```

Nachdem die Einstellungen vorgenommen wurden, sind alle nötigen Konfigurationen für den Apache HTTP-Server abgeschlossen.

Die Komponenten sind jetzt unter folgenden URLs zu erreichen:

MapViewer	→ http://domain.de/mapviewer
ETL (Tomcat)	→ http://domain.de/ETL
APEX	→ http://domain.de/apex

8.6 APEX-Anwendung

8.6.1 Einbinden der Karte

Die Karte wird über Oracle MAPS in die APEX-Anwendung eingebunden. MAPS wird mit dem MapViewer mitgeliefert. Hierfür müssen wir als erstes in den HTML-Header unserer APEX-Seite die MAPS Bibliothek einbinden.

```
<script src="/mapviewer/fsmc/jslib/oraclemaps.js"></script>
```

Als nächstes müssen wir eine Funktion schreiben, die direkt beim Laden der Seite vom Browser aufgerufen wird. Aus Gründen der Übersichtlichkeit werde ich hier nur den relevanten Teil, der für die Darstellung der Karte zuständig ist, aufführen. Der komplette Source-Code kann auf der beiliegenden CD eingesehen werden.

```
<script type="text/javascript">
 function start() {
  // Das MapViewer Theme welches auf der Karte dargestellt
  // werden soll.
  this.stdFOI = 'opensessions';

  // Die URL wo der MapViewer erreichbar ist
  var urlMap  =
    'http://'+document.location.host+'/mapviewer';

  // Eine neue Karte wird erstellt.
  // 10 → Längengrad auf den zentriert werden soll
  // 51 → Breitengrad auf den zentriert werden soll
  // 1  → Zoom-faktor mit dem gestartet wird
  var map = new Map(urlMap, 10, 51, 1, stdFOI);

  // Die Darstellung der Karte starten
  map.start();
}
</script>
```

Um die Darstellung der Karte beim laden der Seite zu starten, muss dem HTML Body Attribut noch folgender Code hinzugefügt werden.

```
onLoad="start();"
```

Der Aufruf der APEX-Seite im Browser (Abbildung 10), zeigt anschließend eine Karte mit den entsprechenden Markierungen der Besucher der zu überwachenden Webseite.

8.6.2 Anzeige der Markierungen

Unterhalb der Karte befinden sich 3 Schalter (Abbildung 10) mit denen die Anzeige der Markierungen beeinflusst werden kann.

offene Session ein/ausblenden

Die auf der Karte angezeigten Session können über diesen Schalter an und ausgeschaltet werden. Je nachdem welche Anzeige gewünscht ist, wird der Layer mit den Markierungen ein oder ausgeblendet.

aktualisieren

Mit diesem Schalter kann der Layer, der die Markierungen anzeigt, aktualisiert werden. Bei Betätigung werden die Daten neu vom MapViewer und damit aus der Datenbank abgerufen.

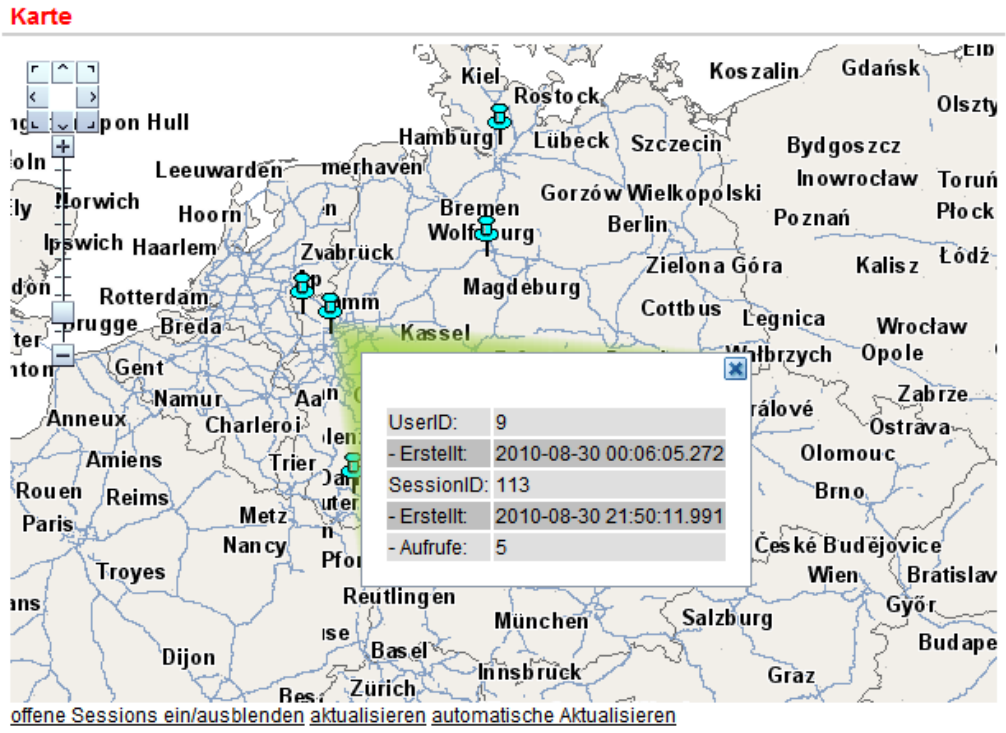

Abbildung 10: Kartendarstellung in der APEX-Anwendung

automatische Aktualisierung

Wird dieser Schalter betätigt, werden die Markierungen automatisch aktualisiert, wenn sich die Daten in der Datenbank ändern. Markierungen werden entfernt, wenn die Session beendet wird und neue Markierungen werden eingefügt, wenn neue Session gestartet werden. Die genaue Funktionsweise beschreibe ich im nächsten Kapitel.

Verwendete Quellen: [9][10][25][29][30][31]

9 Aktualisierung in Echtzeit

Die automatische Aktualisierung der Karte, beim Starten oder Beenden einer Session, gestaltete sich als schwierig. Zum einen lässt der Umstand, dass HTTP ein zustandloses Protokoll ist, eine automatische Aktualisierung nicht zu. Zum anderen sind die Komponenten meiner Anwendung nur darauf ausgelegt, die aktuellen Daten bei einem Aufruf auszuliefern. Da es sich natürlich um ein bekanntes Problem handelt, kursieren verschiedene Lösungen, um dieses Problem zu umgehen.

Eine einfache Variante wäre das sogenannte 'Polling'. Beim Polling wird die Webseite mittels Javascript alle n Sekunden aktualisiert. In meinem konkreten Fall müsste aber nicht die gesamte Webseite neu geladen werden, sondern nur Oracle MAPS aufgefordert werden, den Layer mit den Markierungen zu aktualisieren. Möglich ist mit dem Polling aber nur eine zeitnahe Aktualisierung, je nachdem wie der Zeitraum festgelegt wurde. Tritt eine Änderung der Daten erst kurz nachdem der Layer aktualisiert wurde ein, bleibt die Änderung für die komplette Dauer bis zur nächsten Aktualisierung verborgen. Ein hoher Intervall der Aktualisierung von wenigen Sekunden würde zu dem die Datenbank und auch den MapViewer stark belasten. Zusätzlich wird der Layer bei der Aktualisierung ausgeblendet, und es erscheint ein Ladesymbol was, auch für den Anwender störend ist. Aus den genannten Gründen, habe ich mich gegen ein Polling entschieden.

Letztendlich habe ich mich für eine AJAX-Variante zur Aktualisierung entschieden. Es ist mittels Javascript möglich eine HTTP-Verbindung aufzubauen und über diese Verbindung von einer Änderung der Daten zu erfahren. Aber auch diese Variante bringt einige Probleme mit sich, die ich kurz mit einer Lösung erläutern möchte.

Cross-Site Scripting (XSS)

Um XSS zu vermeiden und damit Sicherheitslücken zu schließen, ist es mit Javascript nur möglich eine HTTP-Verbindung zu einer URL aufzubauen, die der selben Domain inklusive Port angehört, wie die Seite die vom Benutzer aufgerufen wurde. Da alle nach außen hin sichtbaren Komponenten des Systems, durch die Konfiguration des Apache HTTP-Servers als Reverse Proxy, auf der selben Domain und Port ansprechbar sind, ist dieses Problem gelöst.

Einen Auslöser (Trigger) für Änderungen

Änderungen, bei den offenen Session, müssen über eine HTTP-Verbindung abgefragt werden, da Javascript nur HTTP-Verbindungen erlaubt. Leider ist es nicht möglich, eine HTTP-Verbindung zu einem Trigger in der Datenbank aufzubauen, der die Anfrage bei einer Änderung beantworten würde. Das ETL wird aber über einen Webserver angesprochen und der Load-Teil weiß natürlich, wenn neue Daten in die Datenbank geschrieben werden. Es ist also möglich, bei Änderungen am Datenbestand vom Load-Teil benachrichtigt zu werden. Die Lösung für diesen Punkt ist, sich die Informationen über eine Aktualisierung vom Load-Teil mitteilen zu lassen.

Zustandslosigkeit des HTTP-Protokolls

Eine HTTP-Verbindung besteht nur aus einer Anfrage und der darauf folgenden Antwort. Würde jetzt immer wieder eine neue Anfrage gesendet werden, um zu erfahren, ob es eine Änderung der Daten gibt, würde es sich wieder um Polling handeln. Zwar würden bei dieser Lösung die Anfragen für den Benutzer transparent sein, und auch die Datenbank und den MapViewer nicht mehr belasten, dafür aber den Webserver des ETL's. Außerdem würde mit

dieser Methode auch wieder nur eine zeitnahe Aktualisierung je nach Intervall erfolgen. Durch die Angabe von Timeouts der Verbindung auf der Client- und Serverseite, ist es aber möglich eine Verbindung so lange hinauszuzögern, bis eine Änderung der Daten oder der Timeout der Verbindung eintritt. Die Lösung ist die Anfrage durch das Javascript so lange, serverseitig, künstlich hinauszuzögern bis eine Änderung eintritt und den Client dann über die gehaltene Verbindung zu Informieren.

Aufgrund der Probleme und Lösungen, bin ich zu folgender Gesamtlösung gekommen.

Ein aktives Objekt ist für die Entgegennahme der Änderungen vom Load-Teil und Benachrichtigung der eingegangenen Anfragen zuständig. Das Objekt verfügt über eine threadsichere Queue, in die Nachrichten über Änderungen der Daten eingefügt werden. Zusätzlich besitzt das Objekt noch eine weitere threadsichere Queue, in der die Anfragen der Clients eingereiht werden. Gibt es Änderungen an den Daten, prüft das Objekt, ob entsprechende Anfragen für diese Daten vorliegen. Ist dies der Fall, wird der Thread, der die Anfrage gemacht hat, benachrichtigt

Die Anfragen vom Client werden von einem Servlet entgegengenommen. Der Tomcat beauftragt für jede eingehende Anfrage einen Thread, um diese zu bearbeiten. Bei der Bearbeitung wird der Anfrage-Queue des aktiven Objektes eine neue Anfrage eingereiht. Nach der Anfrage legt sich der Thread für einen gewählten Zeitraum (Timeout) schlafen und wartet auf eine Benachrichtigung des aktiven Objektes. Nachdem der Thread über eine Änderung informiert wurde oder der Timeout eingetreten ist, wird der Client informiert und die Verbindung beendet. Bei jeder Antwort wird dem Client ein Timestamp mitgeteilt, den der Client bei jeder Anfrage wieder mitsendet. Anhand dieses Timestamp kann der Server erkennen, wann der Client das letzte mal Informiert wurde und so entscheiden, ob der Client einen aktuellen oder veralteten Datensatz anzeigt.

Auf der Seite des Client wird nach dem aktivieren der automatischen Aktualisierung, eine Verbindung zu dem Server-Servlet aufgebaut. Die Verbindung ist asynchron, damit dieser Vorgang für den Benutzer transparent ist und weiterer Javascript abgearbeitet werden kann. Auch auf der Seite des Client, wird ein Timeout gewählt, der idealerweise minimal höher ist als der serverseitige Timeout. So wird sichergestellt, dass jede Verbindung beantwortet wird. Die Antwort wird vom Client ausgewertet und der Layer mit den Markierungen gegebenenfalls aktualisiert. Nachdem Entgegennehmen der Antwort und dem Abbauen der Verbindung, wird direkt eine neue Verbindung zum Server aufgebaut und auf weitere Aktualisierungen des Datenbestandes gewartet.

Der Client teilt dem Server über HTTP GET Variablen mit, für welche Änderungen er eine Benachrichtigung wünscht und zu welchem Zeitpunkt er das letzte Mal Informationen erhalten hat. Bei dem Zeitpunkt handelt es sich um den Timestamp den der Client bei der letzten Anfrage vom Server geliefert bekommen hat. Die Antwort des Servers erfolgt im JSON-Format.

Chronologisch sieht der gesamte Vorgang folgendermaßen aus:

1. Aktivierung der automatischen Aktualisierung
2. Der Client baut eine Verbindung zum Server auf, um über Änderungen benachrichtigt zu werden.
3. Der Server nimmt die Anfrage entgegen und beauftragt einen Thread mit der Bearbeitung der Anfrage.
4. Der Thread liefert die Anfrage an das aktive Objekt und legt sich anschließend schlafen.
5. Der Load-Teil schreibt neue Daten in die Datenbank und informiert darüber das aktive Objekt.
6. Das aktive Objekt weckt den Threads auf, der auf eine Benachrichtigung für die geänderten Daten wartet.
7. Der Thread verpackt die Informationen im JSON-Format und gibt diese an den Client weiter.
8. Die Verbindung wird auf beiden Seiten getrennt.
9. Der Client wertet die bekommenen Daten aus und aktualisiert den entsprechenden Layer.
10. Der Client startet die Abarbeitung ab Punkt 2. neu.

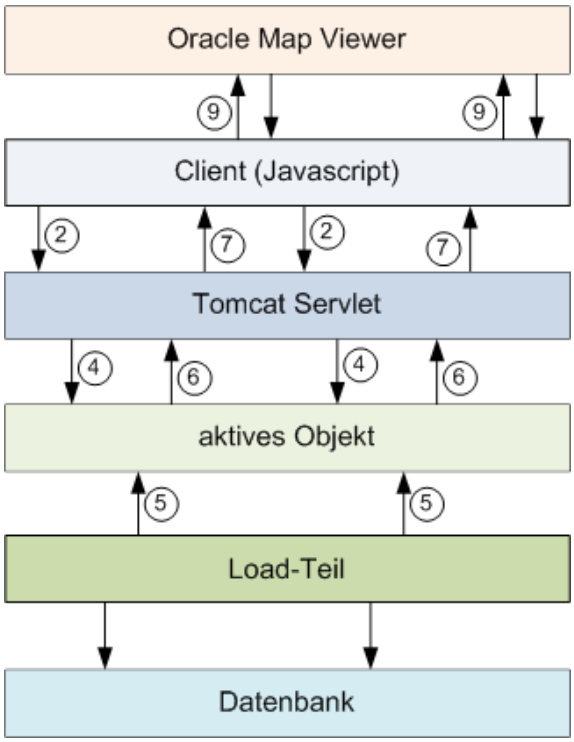

Abbildung 11: automatische Aktualisierung

Mit dieser Lösung ist sichergestellt, dass ein Client immer direkt benachrichtigt wird, wenn sich Daten, die er gerade darstellt, ändern. Er kann direkt eine Aktualisierung durchführen.

Bei meinem System gibt es viele Clients, die die Pixel-Grafik aufrufen, aber nur wenige, die die Auswertung aufrufen. Sollte sich die Anforderung dahingehend ändern, dass viele Clients gleichzeitig die Auswertung betrachten, müssen die Aufgaben des aktiven Objektes auf einen eigenständigen Server ausgelagert werden, der die Verwaltung und das Beantworten der Client-Verbindungen übernimmt und auf weniger Threads verteilt.

JSON-Format

Über das JSON-Format können Daten Sprachen unabhängig ausgetauscht werden. Es handelt sich um ein Format, welches für Menschen leicht zu lesen und schreiben ist. Javascript kann aus einem empfangenen String im JSON-Format direkt ein Objekt erzeugen.

Ein JSON-String den die Clients vom Server bei Änderungen im Datenbestand übertragen bekommen ist folgendermaßen aufgebaut:

```
{reply: '1',date: '1288457285512'}
```

Aus diesem String wird vom Javascript ein Objekt mit den Variablen 'reply' und 'date'

erzeugt. Anhand der Variable 'reply' kann der Client erkennen, ob eine Aktualisierung stattgefunden hat. Ist die Variable gleich 1, hat eine Aktualisierung stattgefunden und der Client aktualisiert die Karte. Wenn die Variable gleich 0 ist, hat keine Aktualisierung stattgefunden und der serverseitige Timeout ist eingetreten. Die Variable 'date' gibt den Zeitpunkt (Timestamp) an, wann der Server den Client das letzte Mal benachrichtigt hat. Der Client sendet den letzten empfangenen Timestamp bei jeder Anfrage mit an den Server. Anhand des Timestamps kann der Server erkennen, ob zwischen der letzten Antwort und der neuen Anfrage Änderungen stattgefunden haben und den Client gegebenenfalls bei einer zwischenzeitlichen Änderung direkt informieren. Mit dieser Methode ist sichergestellt, dass der Client von jeder Änderung zeitnah erfährt.

Verwendete Quellen: [31][32]

10 Fazit

Ziel dieser Arbeit war es, ein System zu entwickeln, mit dem es möglich ist die Zugriffe einer beliebigen Webseite zu überwachen und die örtliche Position der aktiven Benutzer auf einer Karte darzustellen. In der Einleitung habe ich die Kriterien, die die Anwendung erfüllen soll kurz angesprochen und möchte jetzt auf diese Punkte noch einmal genauer eingehen.

Wie in der Aufgabenstellung gefordert, setze ich bei dem System eine Oracle Datenbank für die Speicherung der gewonnenen Daten ein. Die Webseite, auf der die Karte angezeigt wird, ist der Aufgabe entsprechend, als eine Oracle Application Express Anwendung realisiert.

Ein Kriterium ist die Skalierbarkeit des gesamten Systems, bei steigenden Lastaufkommen durch eine Vielzahl von Zugriffen auf die überwachten Webseiten. Es handelt sich also um Zugriffe, die den Pixel-Server und die Datenbank belasten. Durch den Einsatz eines Apache HTTP-Server, der die Last auf n Tomcat Server verteilen kann, ist es möglich weitere Pixel-Server Instanzen dem System zuzuschalten und so die gesamte Last auf mehrere Server zu verteilen. Oracle bietet die Option, mehrere Datenbank zu einem Cluster zusammenzuschalten. Ein Cluster kann eingesetzt werden, um die Verfügbarkeit der Datenbank zu sichern aber auch für die Skalierung. Es ist also auch bei der Datenbank möglich, die aufkommende Last auf mehrere Instanzen zu verteilen. Alle Komponenten des Systems, die von einem erhöhten Lastaufkommen betroffen wären, erfüllen also das Kriterium der Skalierbarkeit. Zusätzlich kann jede Komponente des Systems auf einem eigenen Server betrieben werden.

Ein weiteres Kriterium ist die Unabhängigkeit von einem bestimmten Betriebssystem. Betroffen sind von diesem Kriterium alle Server Komponenten. Also der Apache HTTP Server, der Tomcat Server, die Oracle-Datenbank inklusive dem Application Express und der Oracle MapViewer. Bei der Auswahl der Komponenten habe ich drauf geachtet, dass jede Software dieses Kriterium erfüllt. Der Oracle MapViewer ist eine Java-Anwendung und kann auf allen Systemen betrieben werden, auf den Java lauffähig ist. Alle weiteren aufgezählten Komponenten und Java können auf nahezu allen modernen Betriebssystemen betrieben werden. Untereinader kommunizieren die Komponenten über TCP/IP-Verbindungen. Dieser Umstand erlaubt es jede Komponente auf einem anderen Server mit unterschiedlichen Betriebssystemen zu betreiben und trotzdem ein funktionierendes System zu erhalten. Bei der Entwicklung des Systems habe ich vorrangig ein Windows-Server-Betriebssystem eingesetzt und nach der Fertigstellung das komplette System auf einem Debian-Server installiert. Das Kriterium der Unabhängigkeit vom Betriebssystem ist also erfüllt.

Das letzte Kriterium ist die Darstellung der aktiven Benutzer in Echtzeit auf einer Karte. Um dieses Kriterium zu erreichen, müssen zwei Punkte erfüllt werden. Der erste Punkt ist die direkte Verarbeitung eines Aufrufes, der zu überwachenden Webseite. Durch das einbinden eines Pixels (Tags und Pixel-Lösung) auf der Webseite, erfährt der Pixel-Server umgehend von dem Aufruf kann, die benötigten Daten extrahieren, transformieren und in die Datenbank schreiben. Die ganze Verarbeitung geschieht direkt im Anschluss an den Aufruf. Verzögerungen entstehen nur durch die Laufzeiten der Logik und die Dauer der Kommunikation zwischen den Komponenten. Der zweite Punkt ist die Aktualisierung der Karte, auf der die aktiven Benutzer angezeigt werden. Um diesen Punkt zu erfüllen, ist es nötig die Zustandslosigkeit des HTTP-Protokolls zu umgehen und die Karte zu aktualisieren, wenn Änderungen bei den aktiven Benutzern eintreten. Die Karte wird in einem Browser angezeigt. Es muss also auf eine Technik zurückgegriffen werden, die bei einem Browser von

Haus aus zur Verfügung steht. Ich habe mich für Javascript entschieden. Weiterhin gibt es nur 2 Komponenten in dem System, die wissen, wenn Änderungen am Datenbestand eingetreten sind. Zum Einen die Datenbank und zum Anderen die Pixel-Anwendung auf dem Pixel-Server. Ich habe innerhalb der Pixel-Anwendung eine weitere Komponente eingefügt, die über ein Servlet Anfragen nach Änderungen entgegennimmt und diese erst beantwortet, wenn eine Änderung eingetreten oder eine gesetzte Frist abgelaufen ist. Mittels Javascript wird eine Verbindung zu diesem Servlet aufgebaut und die Karte aktualisiert, wenn die Antwort eine Änderung der Daten bestätigt. Auch bei diesem Punkt treten nur Verzögerungen auf, die von der Laufzeit der Logik und der Dauer der Kommunikation zwischen den Komponenten abhängt. Die Karte wird sofort aktualisiert, wenn eine Änderung bei den aktiven Benutzern der Webseite eintritt und das Kriterium ist damit erfüllt.

Das System erfüllt damit alle Anforderungen, die in der Aufgabenstellung und der gesamten Einführung angesprochen wurden.

Bei dem vorgestellten Design des Systems, kann es bei der Skalierungen zu Schwierigkeiten kommen. Wie erwähnt sitzt die Komponente, bei der Änderungen der aktiven Benutzer abgefragt werden können, in der Pixel-Anwendung. Werden jetzt zusätzliche Pixel-Server Instanzen dem System hinzugefügt, besitzt natürlich jede Instanz eine eigene Komponente, bei der Änderungen abgefragt werden können. Jede Instanz hat aber nur Kenntnis über die Anfragen die bei ihr eintreffen, also die der Lastenverteiler der Instanz zugeordnet hat. Aus diesem Grund ist es wichtig, dass der Lastenverteiler so eingestellt wird, dass er alle Anfragen für eine bestimmtes Pixel (Webseite) immer der selben Pixel-Server Instanz zuteilt. Ansonsten können nicht alle Änderungen dem Client mitgeteilt werden, weil dieser immer nur eine Verbindung zu einem Pixel-Server aufbaut. Außerdem sind der Skalierung Grenzen gesetzt, wenn die Aufrufe einer einzigen Webseite die Kapazitäten einer Pixel-Server Instanz übersteigen.

Das System ist nicht auf eine sehr große Anzahl an Betrachtern ausgelegt, die gleichzeitig die Karte mit den aktiven Benutzern für ein und der selben Webseite automatisch aktualisieren lassen. Jeder Betrachter erzeugt mit seiner Anfrage einen Thread im Pixel-Server, der solange gehalten wird, bis eine Änderung eintritt oder eine Frist abgelaufen ist. Da der Betrachter nach dem beantworten der Anfrage sofort wieder eine neue Verbindung zum Server aufbaut, muss für jeden Betrachter über den gesamten Zeitraum ein Thread eingeplant werden. Bei einer großen Anzahl an Betrachtern, führt es dazu, dass der Pixel-Server keine Ressourcen mehr frei hat um die Pixel Aufrufe oder Anfragen von Betrachtern zu bearbeiten.

An der Stelle der automatischen Aktualisierung der Karte, sollte bei einer Weiterentwicklung angesetzt werden. Eine vom Pixel-Server unabhängige Load-Komponete des ETL könnte über TCP/IP-Verbindungen mit allen Instanzen des Pixel-Servers kommunizieren und über eine HTTP-Schnittstelle Anfragen von den Betrachtern entgegennehmen. Die Load-Komponente könnte dann über Thread-Pools eine große Anzahl an Betrachtern effizient verwalten.

Auch die Kommunikation zwischen Client und Server für die automatische Aktualisierung, kann bei einer Weiterentwicklung optimiert werden. Die jetzige Kommunikation könnte zum Beispiel durch eine Comet-Kommunikation ersetzt werden. Comet nutzt die Eigenschaft von Javascript, dass über eine HTTP-Verbindung eintreffende Daten im XML-Format sofort verarbeitet werden können ohne die Verbindung trennen zu müssen. Durch diese Eigenschaft kann eine persistente Verbindung zwischen Client und dem Server aufgebaut werden.

Eine Weiterentwicklung, kann auch bei der Erhebung der Benutzerdaten ansetzen. Es gibt einige Punkte bei der Erhebung, die von den Einstellungen der Benutzer abhängen. Es wäre zu prüfen ob eine andere Kombination der Verfahren bessere Datenqualität liefert.

Literaturverzeichnis

1: Web-Analytics auf Wikipedia, http://de.wikipedia.org/wiki/Web_Analytics
2: Cookie-Standard RFC 2109, http://tools.ietf.org/html/rfc2109
3: Cookie-Standard RFC 2965, http://tools.ietf.org/html/rfc2965
4: HTTP-Cookie auf Wikipedia, http://en.wikipedia.org/wiki/HTTP_cookie
5: Bericht über ein Urteil des AG München, http://www.schutt-waetke.de/recht_195.htm
6: Google Analytics, http://www.suchradar.de/magazin/archiv/2010/3-2010/google-analytics.php
7: Web-Analytics-Buch.de, http://www.web-analytics-buch.de/
8: Web Server Survey, http://news.netcraft.com/archives/category/web-server-survey/
9: Apache HTTP Server Project, http://httpd.apache.org
10: Apache Tomcat, http://tomcat.apache.org/
11: Microsoft IIS, http://www.iis.net/
12: Lighttpd, http://www.lighttpd.net/
13: Apache HTTP Server auf Wikipedia, http://de.wikipedia.org/wiki/Apache_HTTP_Server
14: Apache Tomcat auf Wikipedia, http://de.wikipedia.org/wiki/Apache_Tomcat
15: MS IIS aus Wikipedia, http://de.wikipedia.org/wiki/Microsoft_Internet_Information_Services
16: Lighttpd auf Wikipedia, http://de.wikipedia.org/wiki/Lighttpd
17: MaxMind, http://www.maxmind.com/app/ip-location
18: Genauigkeit GeoLite City, http://www.maxmind.com/app/geolite_city_accuracy
19: Genauigkeit GeoIP City, http://www.maxmind.com/app/city_accuracy
20: Friedrich Esser, Java 2, 2001, ISBN 3-934358-66-7
21: Friedrich Esser, Java 5 im Einsatz, 2005, ISBN 3-89842-459-6
22: Christian Ullenboom, Java ist auch eine Insel, 2009, ISBN 978-3-8362-1371-4
23: WhereGroup WMS, http://www.wheregroup.com/de/freier_wms_mit_openstreetmap_daten
24: WhereGroup, http://www.wheregroup.com/de/
25: Tutorial Geodaten, http://www.oracle.com/global/de/community/tipps/geo-1/
26: OpenStreetMaps, http://www.openstreetmap.de/
27: OpenLayers, http://openlayers.org/
28: Google Maps, http://maps.google.de/
29: Oracle MapViewer, http://www.oracle.com/technetwork/middleware/mapviewer/
30: Oracle Application Express, http://apex.oracle.com
31: Christian Wenz, JavaScript und AJAX, 2007, ISBN 978-3-89842-859-0
32: Introducing JSON, http://www.json.org/
33: Tutorial MapViewer, http://www.oracle.com/global/de/community/tipps/geo-0/

Tabellenverzeichnis

Tabelle 1: GeoCity - durchschnittliche Genauigkeit weltweit.....21
Tabelle 2: GeoCity - Genauigkeit für Deutschland.....22

Abbildungsverzeichnis

Abbildung 1: Kombination der Verfahren.....13
Abbildung 2: Apache HTTP als Reverse Proxy.....17
Abbildung 3: Pixel-Anwendung mit zwei Modulen.....20
Abbildung 4: ETL-Modul 21
Abbildung 5: Datenbankmodelldiagramm des Datenbankdesigns.....30
Abbildung 6: offene Session.....32
Abbildung 7: Kartensegment und Layer.....34
Abbildung 8: Zusammenspiel der Komponenten.....38
Abbildung 9: Erstellen einer Seite mit dem Oracle Application Express.....40
Abbildung 10: Kartendarstellung in der APEX-Anwendung.....49
Abbildung 11: automatische Aktualisierung.....52

Abbildung 7 ist in Anlehnung an Abbildung 2 aus dem Tutorial 'Karten in Application Express' entstanden [33].

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 08.11.2010

Ort, Datum

Unterschrift