Entwicklerdokumentation

Stand: Juli 2008

Websites:

www.crimestat.de

Ansprechpartner:

Thomas Leimbach	thle0001@stud.fh-kl.de
Natalie Oster	naos0001@stud.fh-kl.de

Inhaltsverzeichnis

1 Einleitung

1.1 Das Projekt CrimeStat

Die Applikation CrimeStat ist das Ergebnis eines Projektes im Rahmen der Veranstaltung Medienkonzeption und Produktion im Sommersemester 2008 an der FH Kaiserslautern am Standort Zweibrücken.
CrimeStat bietet dem Benutzer die Möglichkeit sich Straftaten, welche sich im Raum Berlin ereignet haben, visuell auf einer Karte darstellen zu lassen. Hierzu stehen einige Interaktionsmöglichkeiten zur Verfügung, die dem Benutzer erlauben, weitere Anpassungen vorzunehmen. So ist es möglich einzelne Straftaten-Kategorien auszublenden oder Datumseingrenzungen zu setzen. Weiterhin ist eine gezielte Suche nach Straßen möglich.
Daneben verfügt die Applikation CrimeStat über zahlreiche Statistiken zu einzelnen Bezirken und Straftaten.
Als Informationsquelle dienen die öffentlich zugänglichen Pressemeldungen der Polizei Berlin, welche nach bestimmten Kriterien gefiltert und ausgewertet werden.

1.2 Entwicklerdokumentation

Diese Dokumentation beschreibt die Entwicklung der Applikation CrimeStat. Ausgehend von dem Konzept, über die Implementierung einzelner Komponenten bis hin zu Detailbeschreibungen von bestimmten Algorithmen, soll ein ganzheitlicher Überblick geschaffen werden. Die Dokumentation dient so als Basis für zukünftige Entwickler.
Zum Verständnis dieser Dokumentation sind grundlegende Kenntnisse im Bereich der Programmierung, speziell Java, SQL und JavaScript, erforderlich.

2 Entwicklung

2.1 Aufgabe

Die Realisierung des Projektes kann nur durch eine zuvor festgelegte Definition der Aufgabenstellung durchgeführt werden.
Die Anforderungen an das Projekt CrimeStat lassen sich wie folgt definieren:

- Beschaffung der Pressemeldungen der Polizei Berlins
- Analyse der Pressemeldungen und Informationsextrahierung
- Kategorisierung und Bewertung
- Schaffung einer Visualisierung

2.2 Konzept

Ausgehend von der Aufgabenstellung und den Anforderungen lässt sich die Applikation in zwei Hauptkomponenten unterteilen.
Die erste Komponente stellt die Beschaffung und Auswertung der Pressemeldungen dar. Ziel ist es zunächst, alle notwendigen Informationen zu extrahieren. Dies geschieht mit Hilfe von Java-Code. Zunächst wird eine Verbindung zu der Website der Polizei Berlins hergestellt. Die einzelnen Meldungen werden gespeichert und nach bestimmten Kriterien strukturiert. Die Kategorisierung erfolgt durch einen eigens implementierten Algorithmus, der sich am Bayesschen Filter orientiert. Die Speicherung der Daten erfolgt in einer Datenbank. Zusätzlich ist es notwendig, dass eine ständige Aktualisierung der Daten stattfindet. Dies wurde durch die Verwendung von cronjobs realisiert.
Die zweite Komponente umfasst die Visualisierung der zuvor gewonnenen Informationen.
Hier lässt sich eine Einteilung in zwei Unterbereiche vornehmen. Zum einen die Darstellung auf einer Karte. Dabei kommt die Google Maps API zum Einsatz. Interaktionmöglichkeiten werden durch JavaScript realisiert. Die Darstellung der Statistiken wird mit Hilfe von amCharts realisiert.

2.3 Verwendete Technologie

Java-Komponente:
- Java
- MySQL

Visualisierungskomponente
- HTML
- CSS
- JavaScript

Datenaktualisierung
- cron
- Shell-Scripte

2.4 Systemvoraussetzungen

Die Voraussetzungen für die reine Benutzung der Anwendung lassen sich wie folgt definieren:
- Webbrowser Mozilla Firefox ab v2.0 bzw. Microsoft Internet Explorer ab v6.0
- aktivierte JavaScript-Unterstützung
- Adobe Flash-Plugin ab v.8.0

Für die Entwicklung der Applikation CrimeStat werden neben den oben genannten Voraussetzungen noch die in 2.3 genannten Technologien benötigt. Es empfiehlt sich ebenfalls der Einsatz einer Entwicklungsumgebung wie beispielsweise Eclipse. Die Arbeit mit der MySQL-Datenbank kann durch den MySQL Administrator sowie Query Browser vereinfacht werden.

3. Implementierung der Java-Komponente

3.1 Datenbankaufbau

Die Speicherung der Daten erfolgt in einer MySQL-Datenbank. Um eine nach dem MVC-Pinzip funktionierende Verwendung der Datenbank zu gewährleisten, kommen DTO und DB Klassen zum Einsatz. Die DTO Klassen dienen zur Speicherung der Daten aus der Datenbank. Über die DB Klassen erfolgt der Datenbankzugriff.
Im unten abgebildeten ER-Modell lässt sich der Datenbankaufbau erkennen.

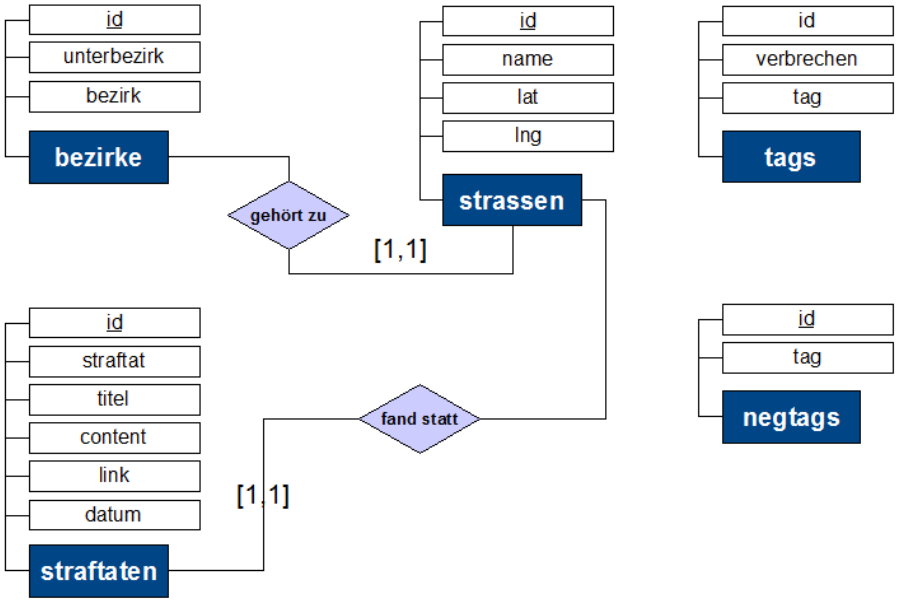

Abb. 3.1, ER-Modell der Datenbank

Die Daten zum Bezirk wurden manuell in die Datenbank eingetragen. Gleiches gilt für die tags sowie negtags. Diese wurden zunächst durch eine Analyse der Pressemeldungen ermittelt.
Die Strassen-Tabelle wurde durch einen eigenen Crawler gefüllt.

3.2 Filterung der Meldungen

Der erste Schritt besteht in der Verbindungsherstellung zur Website der Polizei Berlins. Dies geschieht durch die Klasse *Client*, welche über die Methode *getURLasBuffer()* verfügt. Beim Aufruf der Methode wird die komplette Website in einem String gespeichert.
Der String wird im nächsten Schritt durch die Klasse *MessageFilter* in einzelne Teile aufgesplittet. Die Aufsplittung erfolgt nach bestimmten Pattern.
Nachfolgend ist ein kurzer Code-Ausschnitt abgebildet.

```
public void meldungFilter()
        {
                // Erzeugung des Client-Objektes
                Client client = new Client();
                // Anzahl der Pressemeldungen
                int size = links.size();
                for(int i = 0; i < size; i++)
                {
                        // Speichern der HTML-Seite einer Pressemeldung
                        String meldung = client.getURLasBuffer((String) links.get(i));
                        // Ueberpruefung auf das Suchpattern
                        Matcher mMeldung = pMeldung.matcher(meldung);

                        if(log.isInfoEnabled())
                                log.info("Link: " + links.get(i));
                        while ( mMeldung.find() ) {
                        // Speichern der Meldung in der LinkedList
                                meldungen.add(meldung.substring(mMeldung.start(),
                        mMeldung.end()));

                                if(log.isInfoEnabled())
                                        log.info("Meldung: " +
                meldung.substring(mMeldung.start(), mMeldung.end()));
                        }
                }
        }
```

Abb. 3.2, Auschnitt aus der Klasse MessageFilter

Nachdem eine Strukturierung der Daten erfolgt ist, kann mit der Detailauswertung begonnen werden. Dazu werden aus der Meldung durch die Klasse *LocationAnalyzer* Daten zum Bezirk, Unterbezirk sowie der Straße herausgefiltert. Dies erfolgt ebenfalls durch einzelne Pattern, die jedoch aus den Daten der Datenbank erzeugt werden.

3.3 Kategorisierung der Straftaten

Die Kategorisierung erfolgt nach einem an den Bayesschen Filter angelehnten Algorithmus. Hierzu werden zunächst alle positiven sowie negativen Tags aus der Datenbank geholt. Danach wird ein Array angelegt, welche die Anzahl der Treffer je Kategorie festhält. Für jeden Treffer aus der positiven Tagliste wird an der entsprechenden Stelle im Array der Wert um eins erhöht. Der höchste Wert im Array legt dann am Ende die Kategorie fest. Die negativen Tags dienen dazu, bestimmte Meldungen auszusortieren, die keine Straftaten darstellen. Sobald eine Übereinstimmung mit einem negativen Tag stattfindet, wird die Pressemeldung als keine Straftat gekennzeichnet.
Da der Algorithmus auf zuvor definierten Tags beruht, lässt sich die Genauigkeit durch eine gründliche Anfangsanalyse sowie durch spätere Anpassungen erhöhen.

```
...

// Durchlauf durch die positive Tagliste
while(it.hasNext())
{
        TagsDTO erg = (TagsDTO) it.next();
        // Erzeugen des Patterns zum Pruefen auf ein Tag
        Pattern pTags = Pattern.compile(erg.getTag());

        // Zur Ueberpruefung wird der Content sowie der Header der  Meldung untersucht
        Matcher mTags = pTags.matcher(((String) content.get(i)).toLowerCase() +
        ((String) header.get(i)).toLowerCase());

        // Bei einem Treffer, wird das Feld im Array an der entsprechenden Stelle
// um eins erhoeht
if(mTags.find())
        {
                countArray[erg.getId()-1]++;
        }

        // Hier wird bestimmt, welche Tags, passend zu einer Kategorie,
// die meisten Matches erzielten
        if(maxNumber < countArray[erg.getId()-1])
        {
                maxNumber = countArray[erg.getId()-1];
                if(maxNumber != 0)
                        maxId = erg.getId()-1;
        }
}

...
```

Abb. 3.3, Auschnitt aus der Klasse CrimeAnalyzer

3.4 Paketübersicht

analyze	**CrimeAnalyzer.java** **LocationAnalyzer.java** Diese Klassen dienen zum Herausfiltern des Bezirks / der Straße sowie zur Bestimmung der Straftat
db	**BezirkeDB.java** **BezirkeDTO.java** **DBObject.java** **NegTagsDB.java** **NegTagsDTO.java** **StraftatenDB.java** **StraftatenDTO. Java** **StrassenDB.java** **StrassenDTO.java** **TagsDB.java** **TagsDTO.java** Diese Klassen sind für die Datenspeicherung und Kommunikation mit der DB verantwortlich
filter	**MessageFilter** Hier wird die Pressemeldung aufgesplittet
http	**Client.java** Verbindungsaufbau zur Website und Speichern des Inhaltes in einem String
libs	**commons-codec-1.3.jar** **commons-httpclient-3.0.1.jar** **commons-logging-1.1.jar** **googlemaps.jar** **mysql-connector-java-5.1.6-bin.jar** Libraries die Funktionen wie Logging oder Datenbankverbindungen ermöglichen
main	**ProgramLauncher.java** **StreetConnector.java** **StreetSQLCreate.java** Die Klasse ProgramLauncher führt die Anwendung aus. Die beiden anderen Klassen dienen zur Gewinnung und Speicherung der Straßen.
properties	**db.properties** Property-File zum Speichern der DB Informationen

xml	**CreateCrimeXML.java** **CreateStatisticsXML.java** Diese Klassen sind für die Erstellung der Marker sowie für die Datenquelle der Statistiken verantwortlich

4. Implementierung der Visualisierungs-Komponente

4.1 Layout

Das Layout der Homepage wurde mit HTML und CSS umgesetzt, dabei wurde darauf geachtet, dass der Code browserunabhängig ist. Jedoch ist zu erwähnen, dass die Webapplikation für den Mozilla Firefox optimiert wurde.

Als erstes wurde die Aufteilung der HTML-Seite mit den benötigten Elementen erzeugt. Dabei wurden mehrere div-Blöcke gebildet, die teilweise ineinander verschachtelt sind und mit Ids und Classes ausgezeichnet wurden. Für die Strukturierung einiger Seiteninhalte wurden Tabellen verwendet.

Nachdem der Aufbau der Seite im groben fertig war, wurde damit begonnen die zugehörige CSS-Datei zu erstellen. Hierbei wurden die einzelnen Elemente durch ihre tag-Namen, Ids oder Classes angesprochen und die zugehörigen Attribute vergeben und angepasst. In dieser Datei wurden die Positionierung, Maße, Farben, Schriftarten und –größen sowie Hintergrundbilder angegeben. Vor allem die Positionierung der einzelnen Seitenelemente führte im Hinblick auf die Browser-Kompatibilität zu Problemen. Dieses wurde durch Umstrukturierung der Elemente und Änderung der margin- und padding-Angaben beseitigt.

4.2 Website-Funktionalitäten

Dieser Abschnitt erläutert die Implementierung der einzelnen Funktionen der Website mit Hilfe von JavaScript. Diese sind das Erstellen der Karte, das Einlesen und Erstellen der Marker , die An- und Auswahl der einzelnen Straftaten, das Anzeigen von Straftaten innerhalb eines bestimmten Zeitintervalls, das Zoomen zu einem bestimmten Bezirk sowie die Anzeige eines InfoWindows mit der zugehörigen Meldung.

Erstellen der Karte
Um eine Visualisierung auf einer Karte zu ermöglichen wurde mit der Google Maps API gearbeitet, da diese gut dokumentiert ist und man zahlreiche Beispiele und Foren im Internet findet. Die API ist frei verfügbar. Man muss lediglich einen API-Schlüssel beantragen, der den Zugriff auf Google Maps gewährleistet. Google Maps basiert auf JavaSricpt und bietet viele vordefiniert Funktionen, die verwendet werden können. Dies erleichetert die Einbindung der Karte in eine Website.
Der Vorgang hierfür ist sehr simple. Man bekommt von Google nach Beantragung des Schlüssels bereits einen Beispielcode geliefert, den man so einfach in die HTML-Seite einbinden kann. In diesem Fall wurden die JavaScript-Funktionen in eigene Dateien ausgelagert, so dass in der HTML-Datei nur ein einzelnes div-Tag mit der id

„map“ zu finden ist, das von den Funktionen anschließend angesprochen wird.
Um die Map nun zu erstellen wird im body die Funktion *loadMap()* aufgerufen. Diese erzeugt ein Map-Objekt und fügt einzelne Kontrollen, wie die Suche oder verschiedene Kartentypen, hinzu. In diesem Fall wird eine Karte mit einer lokalen Suche sowie einem ControlPanel und der Möglichkeit den Kartentyp zu wechseln erzeugt. Den dazugehörigen Code können Sie hier sehen:

```
function loadMap() {
  if(GBrowserIsCompatible()) {
   map = new GMap2(document.getElementById("map"));
   map.addControl(new GLargeMapControl());
   map.addControl(new GMapTypeControl());
   map.setCenter(new GLatLng(52.52348, 13.411494), 11);
   // bind a search control to the map, suppress result list
   map.addControl(new google.maps.LocalSearch(), new
GControlPosition(G_ANCHOR_BOTTOM_RIGHT, new GSize(10,20)));
   readData();
  }
 }
```

Einlesen und Erstellen der Marker

Wie man in dem obigen Codeausschnitt sehen kann wird am Ende der Funktion *loadMap()* eine weitere Funktion *readData()* aufgerüfen. Diese Funktion ist dafür zuständig die XML-Datei straftaten.xml, die nach dem Crawlen und speichern der Pressemitteilungen erstellt wird, zu lesen und die Daten herauszufiltern.
Hierfür wird die XML-Datei geöffnet und geparsed. Beim parsen werden die einzelnen Attribute in Variablen abgespeichert. Anschließend wird für jeden Marker die Funktion *map.addOverlay(createMarker(point, html, category, id, datum, content, html2, minContent))* aufgerufen. Diese Funktion erzeugt zum einen den neuen Marker mit den Daten die übergeben werden und platziert diesen Marker anschließend auf der Karte.
Die Methode *createMarker(point, html, category, id, datum, content, html2, minContent)* erzeugt wie bereits erwähnt einen Marker mit den übergeben Daten. Hierfür wird zuerst ein Gmarker-Objekt erzeugt, welches die Koordinaten und das zugehörige Icon übergeben bekommt. Die in der Methode *createMarker()* übergebenen Daten werden zum einen als Markereigenschaften abgespeichert um die Marker zum Beispiel einer Kategorie zuordnen zu können. Zum anderen werden die Daten dem InfoWindow übergeben. Diese Funktion wird später näher erläutert.
Anschließend werden alle Marker in einem Array gespeichert und an die Methode *map.addOverlay()* zurückgegeben.
Ein Codeauszug für die Funktion *createMarker* können Sie hier sehen.

```
function createMarker(point, html, category, id, datum, content, html2, minContent) {
  var marker = new GMarker(point, icons[category] );
   // Store category, name, id, date and icon as marker properties
   marker.category = category;
   marker.id = id;
   marker.date = datum;
   marker.icon = icons[category];
   GEvent.addListener(marker, "click", function() {
    marker.openInfoWindowTabsHtml([new GInfoWindowTab(tab1Label,html), new
GInfoWindowTab(tab2Label,minContent)], {selectedTab:0, maxWidth: maxWidth,
maxHeight: maxHeight, maxContent: html2});
   });
```

```
 gmarkers.push(marker);
 return marker;
}
```

An- und Auswahl der einzelnen Straftaten

Jede Straftat kann mit Hilfe einer Checkbox an- und abgewählt werden. Hierzu wird beim anklicken der Checkbox die Funktion function *boxclick(box, category)* aufgerufen. Diese Funktion überprüft ob die Checkbox an- oder abgewählt wurde und ruft dementsprechend die Funktion *visible.show(category)* oder *visible.hide(category)* auf. Diese Funktionen gehen nach dem selben Prinzip for. Sie gehen das Array mit allen Markern durch und rufen für alle Marker, die der Kategorie zugeordnet sind, entweder die Funktion *show()* oder *hide()* auf. Diese Funktionen sind in der Google Maps API bereits eingebunden und bewirken, dass die Marker angezeigt bzw. ausgeblendet werden.
Hier sehen sie das Codebeispiel für die Funktion *visible.hide()*:

```
var visible= {
 hide: function(category) {
  // Hide all markers of one category
  for(var i= 0; i < gmarkers.length; i++) {
   if(gmarkers[i].category == category) {
    gmarkers[i].hide();
   }
  }
  // Clear the checkbox of a hidden category
  document.getElementById(category).checked = false;
  map.closeInfoWindow();
 }
}
```

Anzeigen von Straftaten innerhalb eines bestimmten Zeitintervalls

Der Benutzer hat die Möglichkeit sich Straftaten innerhalb eines Zeitintervalls anzeigen zu lassen. Hierzu wählt er ein Anfangs und ein Enddatum mit Hilfe einer DatePickers aus und klickt anschließend auf den „anzeigen“ Button. Dieser Button ruft die Funktion *datePicker()* auf. Diese Funktion tut nichts anderes als das Marker-Array durchzugehen bis zum Anfangsdatum und alle folgenden Marker mit der Funktion *show()* anzuzeigen bis das Enddatum erreicht ist. Alle anderen Marker werden mit *hide()* versteckt.

Zoomen zu einem bestimmten Bezirk

Da viele Benutzer der Seite Bewohner Berlins sind, die sich über ihre Nachbarschaft informieren wollen, wurde ein Funktion eingebaut, die es ihnen ermöglicht direkt zu einem Bezirk zu zoomen. Hierfür wurde ein select-Element eingebaut, das alle Unterbezirke Berlins enthält. Bei einer Änderung des select-Elementes wird die Funktion *zoom()* ausgelöst. Diese fragt ab um welchen Bezirk es sich handelt, sucht

aus einem Array die dazugehörigen Koordinaten aus und setzt den Kartenmittelpunkt auf die Koordinaten.

Anzeige eines InfoWindows mit der zugehörigen Meldung

Wie bereits vorhin schon erwähnt werden Daten an ein InfoWindow übergeben . Diese sogenannten InfoWindows erscheinen bei einem Klick auf den Marker innerhalb der Karte. In diesem Fall handelt es sich um TabedInfoWindows, die Informationen zur jeweiligen Straftat.

Um ein solches TabedInfoWindow zu erstellen wird die Funktion *openInfoWindowTabsHtml([new GInfoWindowTab(tab1Label,html), new GInfoWindowTab(tab2Label,minContent)], {selectedTab:0, maxWidth: maxWidth, maxHeight: maxHeight, maxContent: html2})* bezüglich eines Markers aufgerufen. Diese Funktion erstellt ein InfoWindow mit 2 Tabs, denen jeweils ein Tabname und ein Inhalt übergeben wird. Zusätzlich werden noch Angaben darüber gemacht welches Tab beim Öffnen angewählt werden soll und welche maximale Höhe und Breite das Fenster erreichen darf. In diesem Fall wurde noch *maxContent* bestimmt, da die gesamten Pressemitteilungen meistens zu lang sind und nicht in einem kleinen Fenster angezeigt werden können. So hat der Benutzer die Möglichkeit sich die gesamte Meldung in einem größeren Fenster anzeigen zu lassen, wenn sie ihn interessiert.

5. Datenaktualisierung

Ein wichtiges Kriterium für die Funktionalität der Website stellt die automatische Aktualisierung der Daten da. Einmal pro Tag soll eine Abfrage nach neuen Pressemeldungen stattfinden, um den Datenbestand der Website zu erweitern. Um dies zu ermöglichen wird ein cronjob verwendet. Die Definition findet in der crontab statt, die wie folgt aussieht.

```
# Tägliches Crawlen der Pressemeldungen und automatische Aktualisierung
0 0 * * * /home/groups/c/cr/crimestat/update.sh
```

Abb. 5.1, crontab

Die gezeigte crontab führt das Shell-Script update.sh jeden Tag um 0 Uhr aus. Das Shell-Script ruft die Klasse ProgramLauncher auf, welche dann mit dem Crawlen beginnt und die Daten in die Datenbank schreibt.

```
#!/bin/bash

CLASSPATH=/home/groups/c/cr/crimestat/jar/crimestat.jar
java -cp $CLASSPATH main.ProgramLauncher

exit 0
```

Abb. 5.2, Shell-Script update.sh

6 Weiterentwicklung

Die Applikation Crimestat ist mit dem geplanten Funktionsumfang fertig entwickelt. Trotzdem ist es natürlich möglich, weitere Funktionalitäten zu implementieren oder bereits vorhandene zu optimieren.
Da CrimeStat auf Sourceforge.net gehostet ist und somit als OpenSource-Software eingestuft ist, bietet es optimale Voraussetzungen für die Weiterarbeit.