Dynamische Visualisierung von Messwerten aus PEGELONLINE zur Präsentation in digitalen Medien

Projektarbeit IV

vorgelegt am 03.02.2014

von: Torsten Hübel

An der Weiße 18

99310 Arnstadt

Matrikelnummer: E110082IK

Berufsakademie: Berufsakademie Eisenach

Studienbereich: Technik

Studiengang: Informations- und Kommunikationstechnologien

Kurs: IK11

Ausbildungsstätte: Bundesanstalt für IT-Dienstleistungen
Am Ehrenberg 8

D-98693 Ilmenau

Betreuer: Herr Dipl.-Geograph Dietmar Mothes

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API……..	Application Programming Interface
DLZ-IT BMVBS…..	Bundesanstalt für IT-Dienstleistungen im Geschäftsbereich des BMVBS
HTML……..	Hypertext Markup Language
JSON……..	JavaScript Object Notation
PNG…….	Portable Network Graphics
REST……..	Representational State Transfer
SOAP……..	Simple Object Access Protocol
UI……..	User Interface

1 Thema

Über PEGELONLINE werden eine Reihe von aktuellen gewässerkundlichen Messwerten der Bundeswasserstraßen veröffentlicht. Zur Präsentation dienen derzeit klassische Darstellungen von Messwerten. Eine dynamische Darstellung der Werte findet nicht statt. Die Projektarbeit hat das Ziel, neue dynamische Darstellungsmöglichkeiten für die öffentlichen und frei verfügbaren Messwerte von PEGELONLINE zu untersuchen und eine Darstellung prototypisch umzusetzen.

Die inhaltlichen Schwerpunkte sind im Wesentlichen:

- Anforderungen definieren und bewerten
- Bezug der PEGELONLINE-Daten über die vorhandenen öffentlichen Schnittstellen/*Webservices*
- temporäre Datenhaltung zur dynamischen Visualisierung
- Möglichkeiten neuer dynamischer Visualisierungen untersuchen
- eine dynamische Visualisierung auswählen und prototypisch umsetzen
- Darstellung muss dynamisch erfolgen und zur Nutzung in digitalen Medien optimiert sein

2 Einleitung

2.1 Motivation

Im Zeitalter der digitalen Medien stellen Webdienste und Webanwendungen wichtige Informationsquellen für die unterschiedlichsten Arten von Nutzergruppen dar. Das Fachverfahren PEGELONLINE (http://pegelonline.wsv.de/) stellt der Allgemeinheit als Webportal wichtige gewässerkundliche Daten der Binnen- und Küstenpegel, der Wasserstraßen des Bundes, kostenlos zur Verfügung. Der Dienst wird im DLZ-IT BMVBS technisch betreut und auf Servern im Rechenzentrum betrieben.

Für eine Vielzahl der Nutzer, speziell für die außerhalb des gewässerkundlichen Fachbereichs, sind die aktuellen Wasserstände der größte Mehrwert des PEGEL-ONLINE – Webportals. Dabei sind der aktuelle Wasserstand als auch der zeitliche Verlauf der Wasserstände in einem kompletten Gewässer von Interesse. Speziell zu Zeiten des Hochwassers gewinnt das Portal für viele Nutzer stark an Bedeutung. Für die Wasser und Schifffahrt sind Informationen über ein eventuelles Niedrigwasser ebenfalls wichtig, da dies bedeuten kann, dass Wasserstraßen nicht mehr passiert werden können.

Um diese Informationen zugänglicher und übersichtlicher zur Verfügung zu stellen, werden sie in Grafiken visualisiert, was derzeit nur in Form von Ganglinien geschieht.

2.2 Ist-Zustand

Derzeit werden die Pegeldaten auf zwei Arten visualisiert.

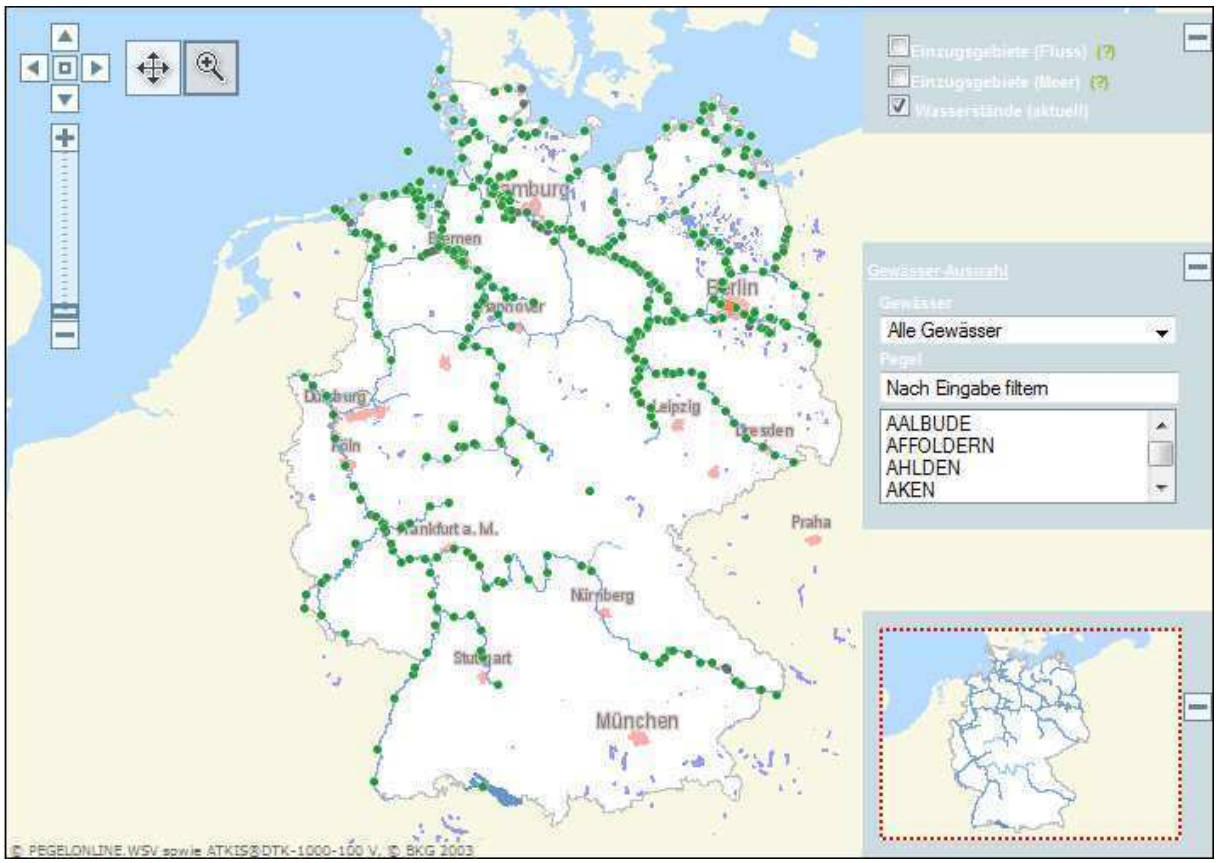

Abbildung 2.2.1 [PO01]Visualisierung und Auswahl über Karte

In der obenstehenden Abbildung ist die Darstellung der Pegel auf einer Karte zu erkennen. Über diese Karte kann der Nutzer Wasserstände direkt per Mausbewegung auf einen Pegel beziehen oder den gewünschten Pegel für mehr Informationen auswählen. Die Punkte färben sich je nach Status (Niedrig-, Normal- oder Hochwassser) des jeweiligen Pegels ein.

Die zweite umgesetzte Möglichkeit zu Darstellung ist die Ganglinie des Wasserstandes. Dabei werden die Messwerte eines Pegels im zeitlichen Verlauf in einem Diagramm dargestellt.

Die nachfolgende Abbildung zeigt exemplarisch die Ganglinie des Pegels Maxau (Rhein) vom 06.12.2013 bis zum 06.01.2014.

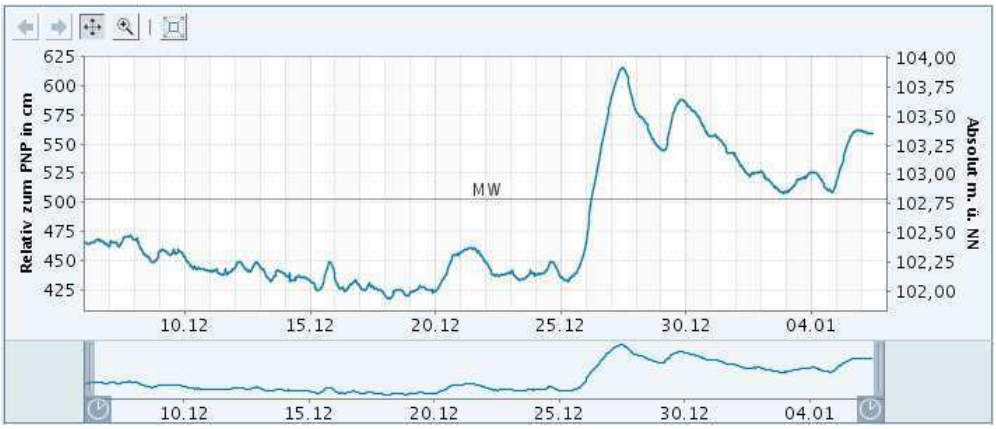

Abbildung 2.2.2 [PO02] Ganglinie des Pegels Maxau (Rhein)

Diese Form der Darstellung bietet schon einen guten Überblick über den Wasserstandsverlauf eines Pegels, jedoch ist es dahingehend limitiert, dass lediglich in einem Diagramm mehrere Pegel durch mehrere Ganglinien dargestellt werden können.

2.3 Zielstellung und Anforderungen

In dieser Projektarbeit sollen neue Möglichkeiten zur Visualisierung dieser Messdaten evaluiert und eine Webanwendung zur Einbindung in ein Webportal erstellt werden.
Die Anforderungen an die Webapplikation werden wie folgt definiert:

1. Die Anwendung soll die benötigten aktuellen Messdaten selbstständig über einen der *Webservices* von PEGELONLINE beziehen.

2. Da es sich um eine Webanwendung handeln wird, die sowie vom Betriebssystem als auch von der Prozessor-Architektur unabhängig ist, soll eine möglichst hohe Kompatibilität zu den verwendeten Internet-Browsern gegeben sein.

3. Die angestrebte Zielgruppe besteht zu großen Teilen aus Nutzern ohne gewässerkundlichen Fachhintergrund. Deshalb sollte die Anwendung selbsterklärend sein und je nach Gewässersituation (Hoch-, Mittel- oder Niedrigwasser) eine intuitive und schnelle Erfassung der Lage ermöglichen.

4. Ebenfalls sollte die Anwendung für den Einsatz in Webseiten sowohl in Bezug auf die Optik, als auch den Funktionsumfang anpassbar und skalierbar sein.

Es soll außerdem untersucht werden, ob eine temporäre Datenhaltung notwendig ist, oder ob die jeweiligen Datenreihen zur Laufzeit und während der Animation direkt von dem Webservice bezogen werden kann.

2.4 Aufbau der Arbeit

In Vorbereitung auf die Erstellung der Anwendung werden im nachfolgenden Abschnitt einige Vorbetrachtungen durchgeführt. Darin wird neben der Prüfung und Auswahl einer geeigneten Webschnittstelle auch die Sichtung geeigneter Programmier-Plattformen abgehandelt. Im Anschluss werden Möglichkeiten zur sinnvollen dynamischen Darstellung der Messwerte untersucht.
Im Abschnitt 4 „Umsetzung“ wird der Aufbau und die Funktionsweise der Anwendung, sowie deren Erstellungsprozess mit den zu beachtenden Besonderheiten beschrieben.
In dem darauf folgenden Abschnitt 5 „Ausblick“ wird die entstandene Anwendung kritisch auf technische und konzeptionelle Fehler untersucht, sowie die Möglichkeit zur Integration in eine Webseite aufgezeigt.
Die Zusammenfassung bildet den Abschluss der Arbeit und enthält einen kurzen Rückblick.
Eine umfangreiche Sammlung an Begriffsklärungen ist im Glossar aufzufinden. Darin befindliche Begriffe sind durch deren kursive Schriftart gekennzeichnet. Im Abkürzungsverzeichnis werden die verwendeten Abkürzungen erläutert.

3 Vorbetrachtungen

3.1 Schnittstellen zum dynamischen Bezug der PEGELONLINE – Daten

PEGELONLINE stellt seinen Nutzern die Messwerte auf der Webseite www.pegelonline.wsv.de zur Verfügung. Außerdem bietet es noch eine Reihe an Schnittstellen , sogenannten Webservices, an. Mit Hilfe dieser Webservices können andere Applikationen die Pegeldaten beziehen und weiter nutzen .

Neben geografischen Informationsdiensten, die es erlauben gewässerkundliche Daten in eine Kartenanwendung zu integrieren, stellt PEGELONLINE auch auf *SOAP* und REST basierte Webservices an.

Mit Hilfe des weit verbreiteten **S**imple **O**bject **A**ccess **P**rotocol´s können Daten zwischen dem Webdienst und einer Anwendung auf Basis von XML ausgetauscht werden. SOAP ist ein standardisiertes Protokoll, für das in den meisten Programmiersprachen eine Implementierung existiert. Dank der Standardisierung des Protokolls müssen sich Server und Client vor der Datenübertragung nicht über eine Datenstruktur einigen, jedoch erfordert die Implementierung eines SOAP Clients umfangreiche Programmierkenntnisse und ist je nach Programmiersprache unterschiedlich anspruchsvoll.

in Datenaustausch auf Basis eines REST-Dienstes wird von PEGELONLINE ebenfalls zur Verfügung gestellt. Der **Re**presentational **S**tate **T**ransfer ist nicht vollständig standardisiert. Somit muss der Client wissen, welche Datenstruktur er vom Server zu erwarten hat. Der Client hat die Möglichkeit durch ansprechen einer spezifischen URL eine Funktion auf dem Server auszuführen, welche dem Client wiederum entsprechende Daten zur Verfügung stellt. Der PEGELONLINE REST-Dienst bietet Messwerte und allgemeine Daten von Pegeln in Form eines Javascript Objektes (JSON) zum download an.

Durch Ansprechen der URL:

```
http://pegelonline.wsv.de/webservices/rest-
api/v2/stations/BONN/W/currentmeasurement.json
```

wird beispielsweise die jeweils aktuelle Messung des Pegels „Bonn“ in Form eines JSON Objektes an den Client übermittelt:

```
{
  "timestamp": "2014-01-07T07:00:00+01:00",
  "value": 411.0,
  "trend": 1,
  "stateMnwMhw": "normal",
  "stateNswHsw": "normal"
}
```


Abbildung 3.1.1 [PO03] JSON - Objekt

Dieses Objekt kann dann im Programmcode weiter verarbeitet werden. Bei der Verwendung von Javascript bieten JSON Objekte den Vorteil, dass diese nicht erst von der Anwendung *geparsed*, also analysiert werden müssen, sondern direkt im Code weiter verwendet werden können.
Somit stellt der REST-Webservice eine leichtgewichtige und performante Möglichkeit dar um Daten zwischen einem Client und einem Webserver auszutauschen und eignet sich gut für die Verwendung in mobilen und Javascript-Anwendungen.
Der PEGELONLINE REST-Dienst kann weiterhin Ganglinien im *PNG-Format* bereit stellen.
Die folgende Ganglinie wird beispielsweise durch Ansprechen der URL

„http://pegelonline.wsv.de/webservices/rest/api/v2/stations/BONN/W/measurements.png„

zurück gegeben:

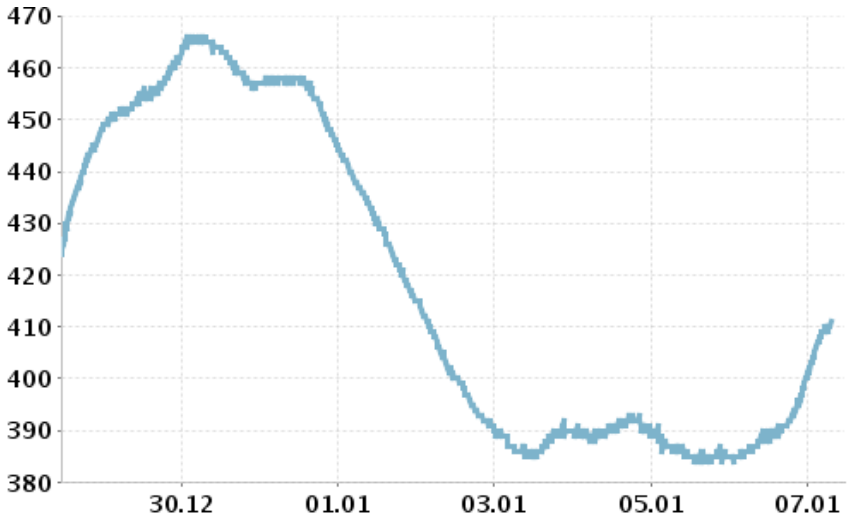

Abbildung 3.1.2 [PO04] Ganglinie von REST-Service

Da dieses Feature für die dynamische Visualisierung der Wasserstände keine Bedeutung hat, wird es in der zu erstellenden Anwendung nicht benötigt.

Webanwendungen werden häufig mit HTML und *Flash* oder *Javascript* aufgebaut. Da sich diese Skriptsprachen hervorragend zur Kommunikation mit REST – Webservices und Verarbeitung von JSON – Objekten eignen, ist die Verwendung des REST Dienstes für diese Anwendung vorzuziehen.

3.2 Die Entwicklungsumgebung

3.2.1 Sichtung geeigneter Plattformen zur dynamischen Darstellung von Messwerten auf Webseiten

Grundlegend sollte die Darstellung der Wasserstände in Form eines animierten Diagramms erfolgen. Zum Erstellen von statischen Diagrammen gibt es bereits einige Webapplikationen auf dem Markt. Diese Anwendungen bieten jedoch nur statische Bilder oder nur unzureichende Möglichkeiten der Animation von Datenreihen und Interagierbarkeit.

Um die geforderte Funktionalität umsetzen zu können wird eine Programmierschnittstelle oder eine Erweiterung zu einer web-tauglichen Scriptsprache wie Flash oder Javascript benötigt.

Eine auf Adobe Flash basierte Lösung bietet das freie Projekt Open Flash Chart ([OFC01]), das es erlaubt mit Diagrammen zu interagieren und diese auch über Umwege zu animieren.

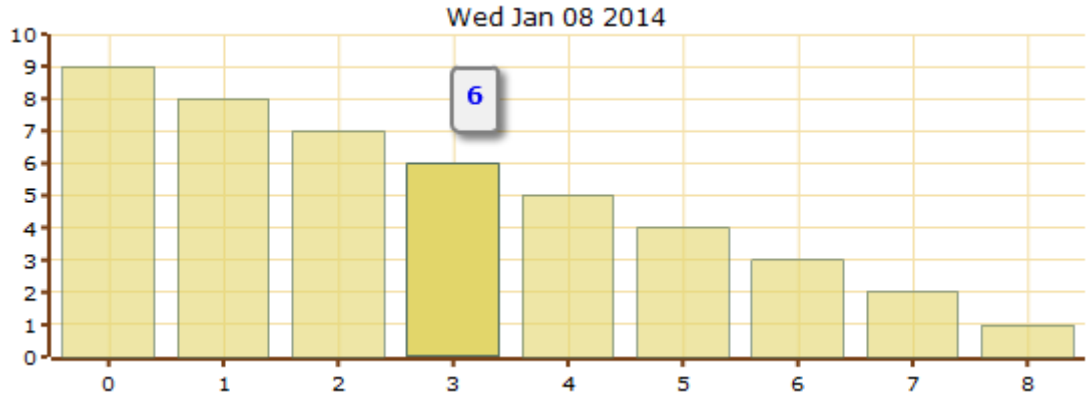

Abbildung 3.1.2 [OFC02] Beispieldiagramm Open Flash Chart

Die Verwendung von Adobe´s Flash ist jedoch kritisch zu betrachten und die Anpassbarkeit dieser Lösung ist begrenzt.

Einige andere Erweiterungen zum dynamischen Erstellen von Diagrammen stützen sich auf das weit verbreitete JavaScript.

Eine bekannte API stellt dabei die Flot Chart API ([FC01]) dar. Flot ist eine Bibliothek für das JavaScript Framework *JQuery ([JQ01])* und ermöglicht das Erstellen von interagierbaren Diagrammen. Dank der Verwendung von JavaScript ist eine Kommunikation zu anderen Programmteilen und Schnittstellen wie dem REST-Webservice ein-

fach zu implementieren. Eine Animation von Datenreihen ist mit dieser API zwar nicht möglich, jedoch können die angezeigten Daten ausgetauscht werden und somit die Messwerte zu den einzelnen Zeitpunkten nacheinander dargestellt werden. Durch externen Code, der die Datenreihen zeitlich versetzt austauscht, kann die Animation implementiert werden.

Google bietet mit seiner Google Charts API die umfangreichste Programmierschnittstelle zum dynamischen Erstellen von Diagrammen an. Mit über zwanzig verschiedenen Diagrammtypen und vielen Möglichkeiten zum Interagieren sowie hinzufügen von Steuerungselementen bildet Google Charts eine Alternative zu den vorher genannten Lösungen. Die Nutzung von Google Charts ([GC01]) ist wie bei den anderen Programmierschnittstellen kostenlos und frei.
Ein zeitliches animieren von Datenreihen ist ebenfalls nicht vorgesehen, jedoch ist ein Wechsel der dargestellten Datenreihen mithilfe einer konfigurierbaren Übergangsanimation implementierbar.

Microsoft bietet auf Basis von *Silverlight* auch eine Option zum dynamischen Erstellen von Diagrammen an. Da Silverlight derzeit noch nicht auf mobilen Geräten verwendet werden kann und die Verbreitung insgesamt relativ gering ([HDV01]) ist, werden keine weiteren Betrachtungen dahingehend ausgeführt.

3.2.2 Begründung und Auswahl eines Systems

Nachdem die Silverlight-Lösung bereits aus Kompatibilitäts-Gründen ausgeschieden ist, stehen nun noch Open Flash Chart, Flot Chart und Google Charts zur Auswahl.

Die Verwendung von Flash war in der Vergangenheit im Internet weit verbreitet. Seitdem sich mobile Geräte wie Smartphones und Tablets immer mehr als internetfähige Geräte etabliert haben, ist ein ständiger Rückgang der Integration von Flash-Inhalten in moderne Webseiten zu verzeichnen. Das liegt unter anderem daran, dass die meisten Smartphones und Tablets keine Flash-Kompatibilität aufweisen.
Deshalb ist es aus heutiger Sicht nicht sinnvoll eine Flash-basierende Anwendung zu entwickeln und zu publizieren.
Die Entscheidung fällt also zwischen den beiden JavaScript – Lösungen Google Charts und Flot Chart.
Aufgrund der größeren Vielfalt was Design, Diagrammtypen und Anpassbarkeit angeht wird in diesem Projekt die Google Charts API verwendet. Ein weiteres Entscheidungskriterium war, dass sich die Übergänge beim Wechsel der Datenreihen animieren lassen und sich somit die Bedienung für den Anwender flüssiger erscheint.

3.3 Möglichkeiten zur sinnvollen Darstellung der Messwerte und Auswahl einer Variante

Durch die Verwendung von Google Charts gibt es eine Reihe von unterschiedlichen Möglichkeiten zur Darstellung von Daten, vorher sollte man sich Gedanken darüber machen was überhaupt dargestellt werden soll.

Für den Nutzer wäre es jedoch auch die Animation des zeitlichen Verlauf mehrerer Wasserstände entlang eines Flusses im Querschnitt interessant. Der nachfolgende Entwurf soll verdeutlichen, wie eine Welle entlang des Flussverlaufs abwärts verläuft..

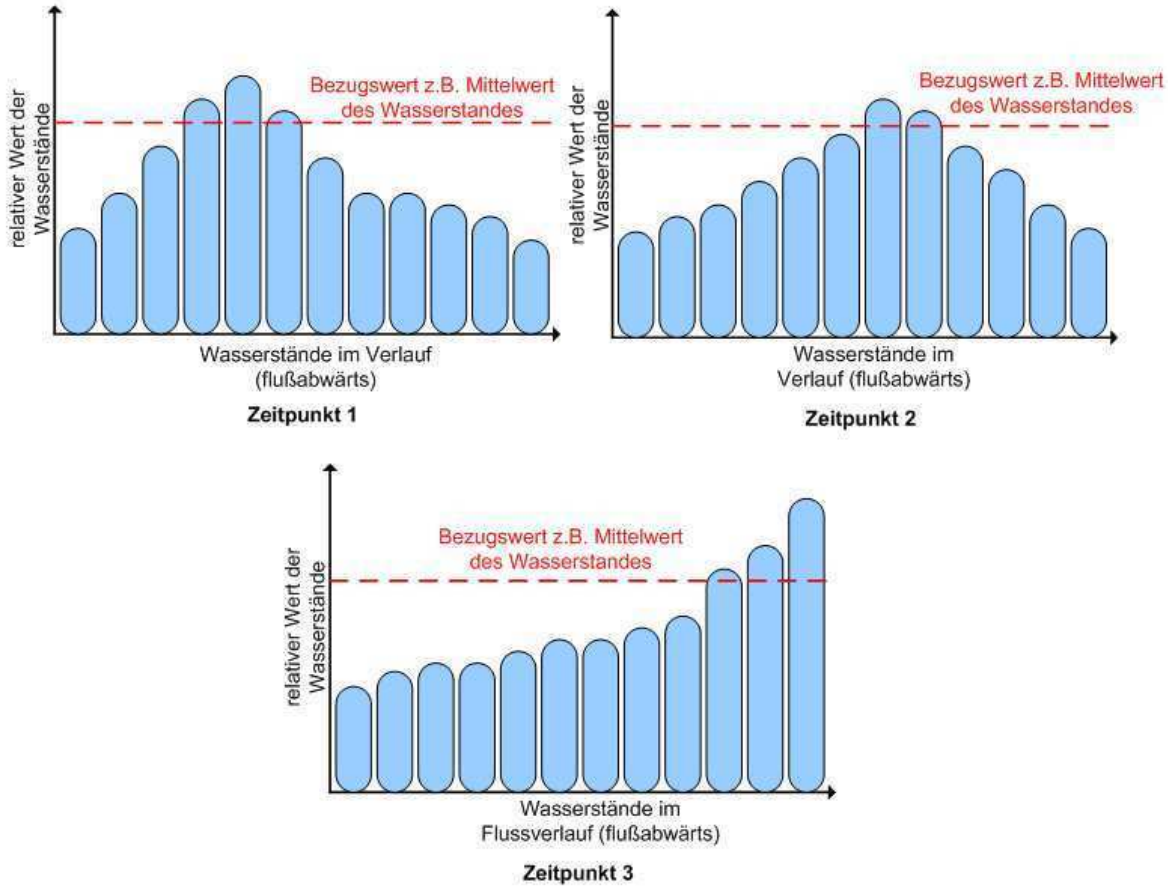

Abbildung 3.4.1 Entwurf Darstellung Wasserstand

Die von den Pegelstationen gemessenen Wasserstandswerte stützen sich jeweils auf einen anderen Bezugspunkt. Daher ist es sinnvoll die Messwerte auf einen individuellen Vergleichswert zu beziehen. Die nachfolgende Tabelle stellt dies beispielhaft dar und dient zur Verdeutlichung des Sachverhalts.

Pegel	Wasserstand	mittlerer Wasser-stand	darzustellender Wert
Pegel A	750 cm	600 cm	150 cm
Pegel B	550 cm	390 cm	160 cm

So wird beispielsweise ersichtlich, wie sich die Wasserstände im Verhältnis zu deren jeweiligen Wasserstandsmittelwerten im Verlauf der letzten Tage entwickelt haben. Bei einem Hochwasser ist damit zu erkennen wie sich die kritischen Wasserstände im zeitlichen Verlauf flussabwärts bewegen (siehe Abbildung 3.4.1).

Wie bereits im Abschnitt 2.2 besprochen wurde, bietet PEGELONLINE schon jetzt die Möglichkeit der Darstellung von einer oder mehrerer Ganglinien in einem Diagramm. Dabei wird jedoch immer nur der Wasserstand als Funktion der Zeit dargestellt.

Die naheliegendste Variante wäre eine Abbildung in Form eines klassischen Balken-diagramm wie es in dem vorangegangenen Entwurf abgebildet ist. Da Google Charts noch deutlich mehr Diagrammtypen anbietet, werden im Folgenden passende Typen analysiert.

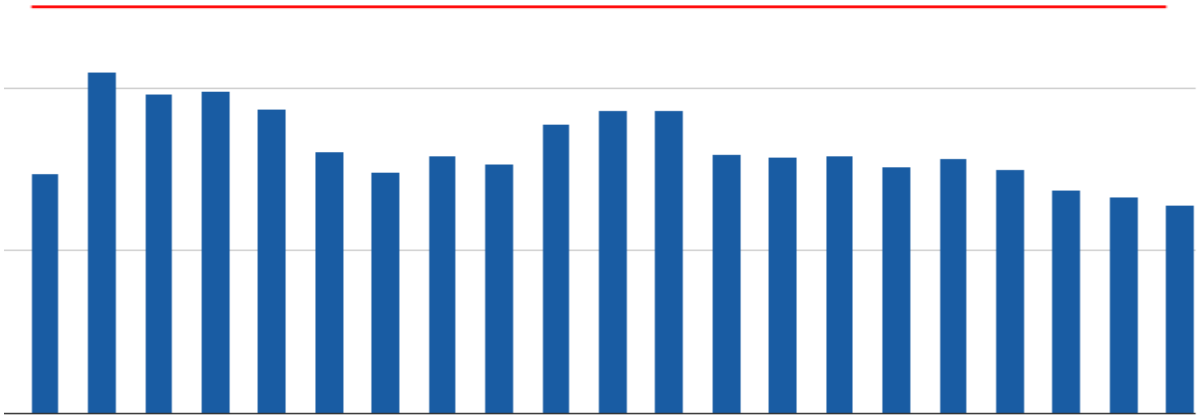

Abbildung 3.4.2 Combo-Chart

Die einfache Darstellung als Balken oder Stabdiagramm erfolgt im sogenannten Combo-Chart (siehe Abbildung 3.4.2). Dabei werden die einzelnen Wasserstände als Balken getrennt voneinander dargestellt. Die rote Linie stellt dabei den Vergleichswert dar.

Abbildung 3.4.3 Line-Chart

Das Line-Chart Diagramm (siehe Abbildung 3.4.3). verbindet jeweils die oberen Punkte der Wasserstände. Da die Anzahl der darzustellenden Pegel jedoch begrenzt ist, wirkt die Linienführung sehr abgehakt und optisch wenig ansprechend. Das Area-Chart entspricht dem Line-Chart und füllt zusätzlich dazu die Fläche unter der Linie aus und eignet sich deshalb auch eher weniger zur Darstellung der Wasserstände.

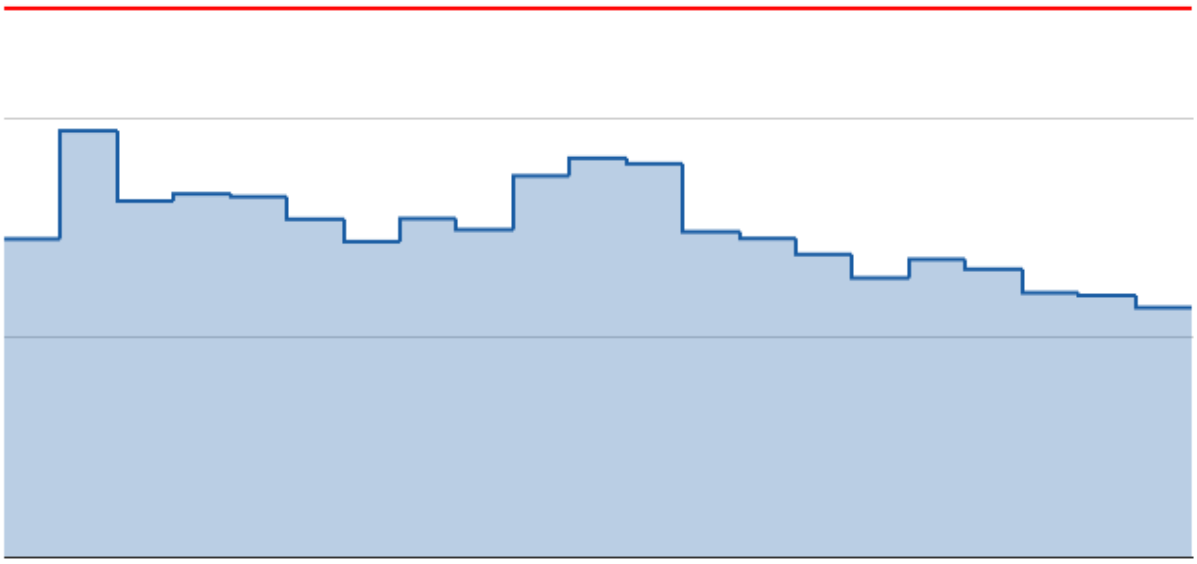

Abbildung 3.4.4 Stepped Area Chart

Eine Mischung aus Area-Chart und Combo-Chart(siehe Abbildung 3.4.2) bietet das Stepped Area Chart (siehe Abbildung 3.4.4). Es Verbindet die maximalen Werte als Balken mit einer Linie und füllt die Fläche darunter aus.
Dieser Diagrammtyp eignet sich gut für die Bewältigung der gegebenen Aufgabe, da es optisch am besten vermitteln kann, dass es sich bei den angezeigten Daten um Wasserstände handelt und wurde aus diesem Grund für die Verwendung in diesem Projekt ausgewählt.

4 Umsetzung

Aus den Voruntersuchungen haben sich für die Umsetzung der Aufgabe folgende technische Werkzeuge als die geeignetsten ergeben:

- Verwendung der Google-Charts API zur Darstellung der Messwerte
- Dynamischer Bezug der Daten per PEGELONLINE-REST-Webservice
- Steuerung der Anwendung mit externen Javascript-Code

Unter Verwendung dieser Mittel ist eine Anwendung zur dynamischen Visualisierung der Wasserstände des Rheins entstanden.
Da für diese Anwendung die Benutzeroberfläche ein sehr wichtiger Aspekt ist, werden im nachfolgenden Abschnitt vorerst die Verwendung und der Aufbau der Benutzeroberfläche abgehandelt um danach auf technische Details einzugehen.

4.1 Aufbau der Benutzeroberfläche und Funktionsweise der Anwendung

Die programmierte Anwendung bietet dem Nutzer die Möglichkeit sich den Verlauf der Wasserstände des Rheins in einem Zeitraum darstellen zu lassen. Dabei werden die einzelnen Wasserstände im Bezug zu einem charakteristischen Wert, wie beispielsweise dem mittleren Wasserstand, zu einem Zeitpunkt oder durch eine zeitliche Animation dargestellt.

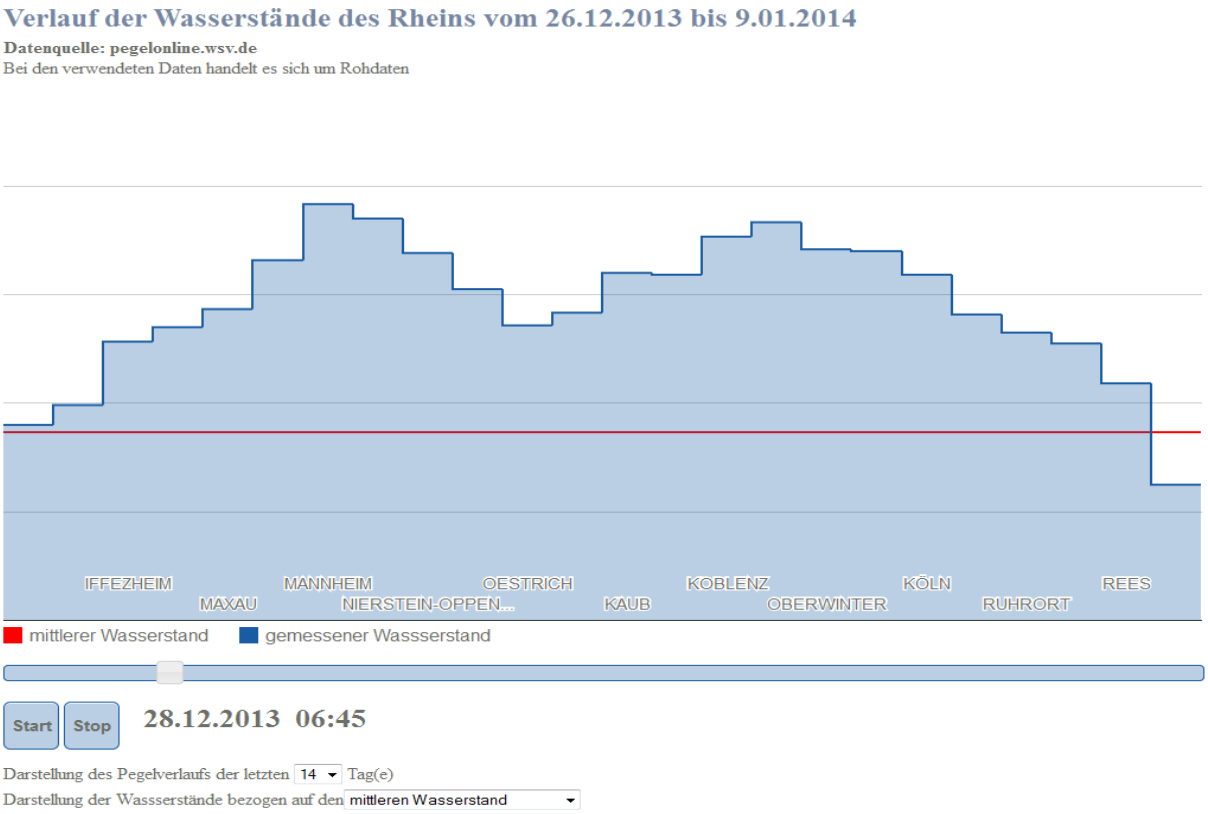

Abbildung 4.1.1 Darstellung der gleichen gemessenen Wasserstände bezogen auf unterschiedliche hydrologische Kenngrößen

Wie in Abbildung 4.1.1 zu erkennen ist, werden die Messstellen des Rheins von links nach rechts flussabwärts dargestellt. Um die Übersichtlichkeit zu bewahren wird nicht jeder Messpunkt beschriftet, sondern je nach Skalierung nur circa jeder Zweite. Neben dem eigentlichen Diagramm befinden sich noch einige Steuerungselemente auf dem User Interface. Der Slider unter dem Diagramm wird per Javascript von der JQuery Bibliothek dynamisch erstellt und dient gleichzeitig als Fortschrittsanzeige der Animation und als Möglichkeit zum Auswählen eines spezifischen Zeitpunktes.

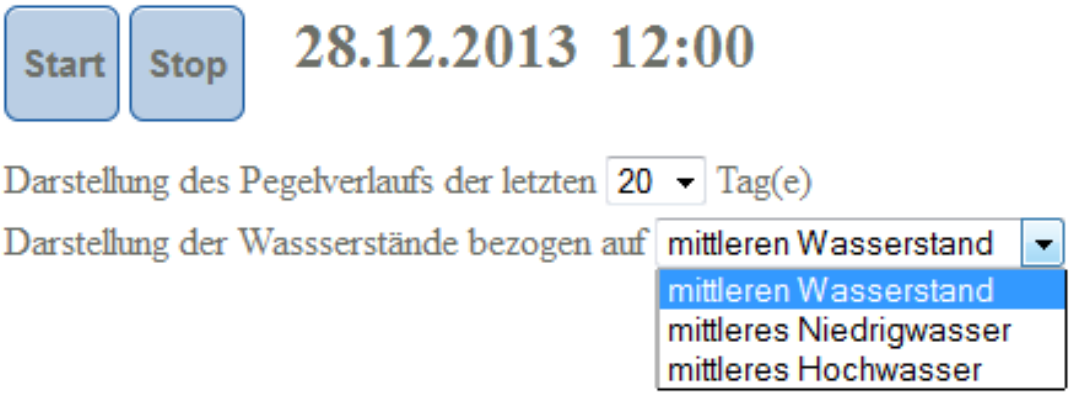

Abbildung 4.1.2 Steuerungselemente

Des Weiteren kann mit Hilfe der Start- und Stop-Buttons die Animation gestartet, angehalten oder gestoppt werden.
Die Auswahl der darzustellenden Zeitspanne erfolgt ebenso über eine Auswahlliste wie die Wahl der Bezugsgröße. Das ist sinnvoll, da je nach allgemeiner Hoch- beziehungsweise Niedrigwasser-Lage eine andere Ansicht den tendenziellen Verlauf der Wasserstände am besten verdeutlicht. In der nachfolgenden Abbildung wird zum selben Zeitpunkt die Darstellung derselben Wasserstände, mit unterschiedlichen Bezugsgrößen verglichen. Daran wird die Notwendigkeit zur Auswahlmöglichkeit dieses Wertes deutlich.

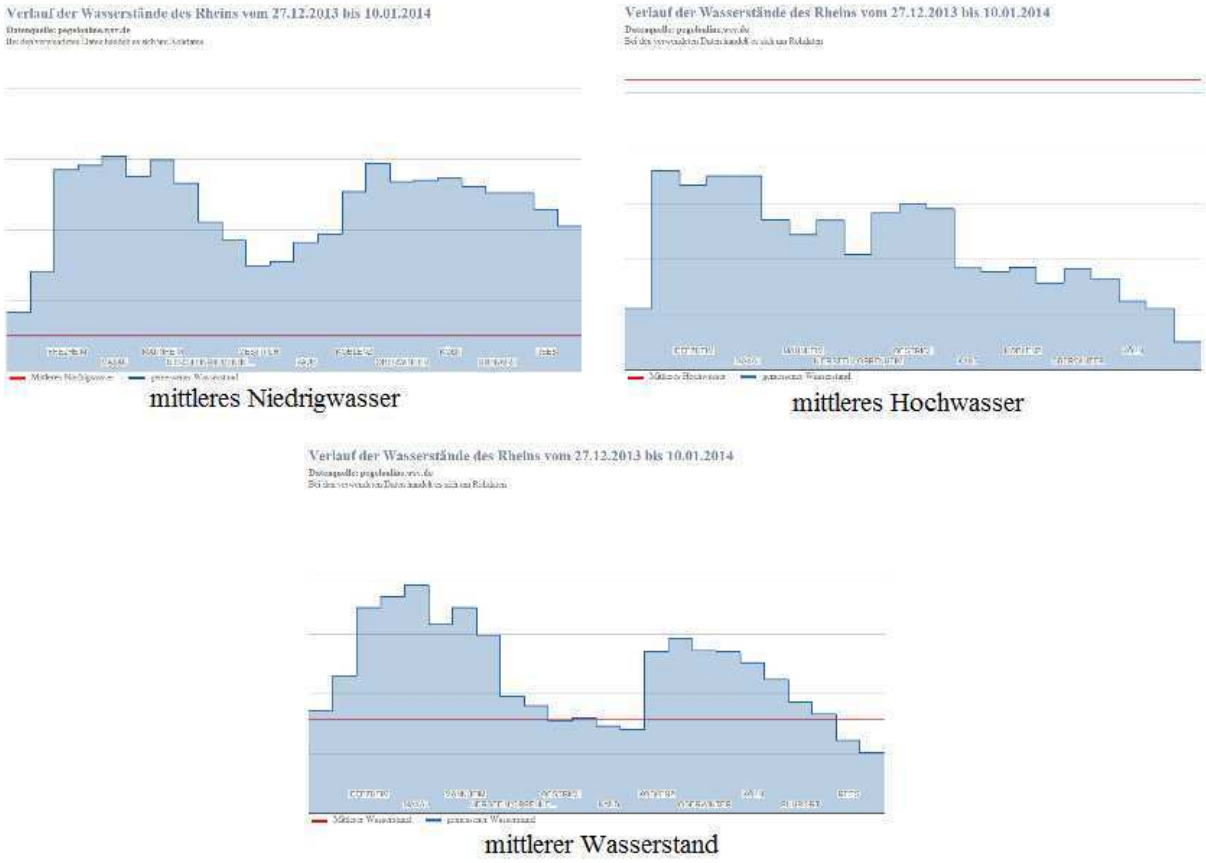

Abbildung 4.1.3 Darstellung mit unterschiedlichen Bezugsgrößen

Zum Zeitpunkt der Aufnahme wäre die Darstellung mit Bezug auf das mittlere Niedrigwasser oder das mittlere Hochwasser nicht repräsentativ, da keine Tendenz der Unter- beziehungsweise Überschreitung dieser Bezugswerte zu erkennen ist.

Nachdem die Funktionsweise und der Aufbau der Anwendung abgehandelt wurden, wird im Folgenden auf die technische Umsetzung eingegangen.

4.2 Bezug und Verarbeitung der Messwerte

Das entstandene dynamische Diagramm lebt in erster Linie von seinen darzustellenden Daten. Da diese Wasserstands-Messwerte nicht statisch sind, sondern während der Laufzeit bezogen werden, ist die Datenbeschaffung und Haltung aus technischer Sicht von großer Bedeutung.

4.2.1 Bezug der Messwerte per REST – Webservice

Im Abschnitt 3.1 „Schnittstellen zum dynamischen Bezug der PEGELONLINE – Daten“ hat sich der REST-Webservice als geeignete Schnittstelle zum Bezug der Messwerte heraus gestellt.
Dabei stellen die PEGELONLINE – Webserver nach Ansprache von definierten URLs durch den Client JSON-Objekte mit den entsprechenden Daten bereit. Die Abfragemöglichkeiten sind dabei sehr umfangreich und in der öffentlich verfügbaren API Dokumentation ([PO05]) hinterlegt.
Da vorerst nur der Verlauf des Rheins dargestellt werden soll, prüft die Anwendung im ersten Schritt bei jeder Ausführung, welche Messstationen im Verlauf des Rheins verfügbar sind. Dies erfolgt durch das Ansprechen der URL:

```
http://www.pegelonline.wsv.de/webservices/rest-
api/v2/stations.json?waters=RHEIN&includeTimeseries=true&includeCharacteris
ticValues=true
```

Der Webservice gibt daraufhin die Liste aller Pegel am Rhein mit deren charakteristischen Werten, wie z.B. den mittleren Wasserstand zurück.
Diese Liste wird vom Programmcode durchgearbeitet und nur valide Pegel zur Weiterverarbeitung berücksichtigt.
Valide Pegel sind diejenigen, die Informationen zu dem eingestellten Wasserstandsbezugswertes enthalten. Möchte der Nutzer seine angezeigten Messwerte also beispielsweise auf die mittleren Wasserstände beziehen, so werden nur die Pegel berücksichtigt, bei denen dieser Wert angegeben ist.

Von den berücksichtigten Pegeln werden dann mit jeweils einer Anfrage die Wasserstände eines spezifizierten Zeitraums bezogen.

```
http://www.pegelonline.wsv.de/webservices/rest-api/v2/stations/06b978dd-
8c4d-48ac-a0c8-2c16681ed281/W/measurements.json?start=P14D
```

Mit der obenstehenden URL spricht die Anwendung die Messwerte des Pegels “Rheinweiler“ an und bekommt alle gespeicherten Wasserstände der letzen 14 Tage als JSON Objekt zurück. Da alle 15 Minuten eine Messung erfolgt, werden somit pro Tag 96 Messwerte vom REST-Webservice zurückgegeben.

Der Bezug der Daten findet vor der eigentlichen Ausführung, also während der Initialisierungsphase der Anwendung statt. Eine andere Möglichkeit wäre, die Daten während der Ausführung zu beziehen, da somit die Ausführungsgeschwindigkeit aber von der Bandbreite der Internetverbindung abhängig wäre, wurde sich gegen diese Methode entschieden.

4.2.2 Interne Datenhaltung und Verarbeitung der Messwerte

Da der Bezug der Daten vor der eigentlichen Ausführung stattfindet, müssen die Messdaten in internen Datenstrukturen zwischengespeichert werden.
Das Javascript könnte theoretisch auch direkt mit den erhaltenen JSON-Objekten arbeiten, das mit Google Charts erstellte Diagramm erwartet jedoch eine tabellenähnliche und keine objektorientierte Datenstruktur.
Deshalb werden die einzelnen JSON Objekte zu einem dreidimensionalen Array konsolidiert.

Zeitpunkt 1												Zeitpunkt 2											
Station1				Station 2				Station 3				Station 1				Station 2				Station 3			
P1	P2	P3	P4	P1	P2	P3	P4	P1	P2	P3	P4	P1	P2	P3	P4	P1	P2	P3	P4	P1	P2	P3	P4

P1 - Zeitpunkt P2 - Name der Station P3 - Messwert P4 - Bezugswert

Abbildung 4.2.2 Interne Datenstruktur

Die obenstehende Abbildung verdeutlicht wie die Daten in der Anwendung temporär gespeichert werden. Dabei steht die erste Dimension für alle zu berücksichtigende

Zeitpunkte, die zweite für alle vorhandenen Messstationen und die dritte für die eigentlichen Werte.
Um die Performance und Übersichtlichkeit zu verbessern wurde festgelegt, dass nur zwölf Messwerte pro Tag berücksichtigt werden, somit werden nur Daten im Abstand von zwei Stunden angezeigt.

Das Google Charts Diagramm hat seine eigene Datenhaltung hinterlegt, die neben den eigentlichen Messdaten auch Metadaten, wie *Tooltips* und Beschriftungen enthält.
Deshalb müssen die Daten aus der Anwendungs-Datenstruktur zu dem Diagramm übertragen werden.

Die Google-Charts API erlaubt lediglich das Einlesen einer einzelnen Datenreihe. Somit müssen jeweils zum Start und an jedem Zeitpunkt der Animation die Datenreihen mit den Messwerten zum jeweiligen Zeitpunkt eingelesen werden.

4.3 Dynamische Animation der Daten

Eine Möglichkeit des zeitdiskreten Tauschens der aktuellen Datenreihe ist von der Google-Charts API nicht vorgesehen. Ein Wechsel aller angezeigten Daten ist jedoch möglich und der Übergang kann animiert. Somit können die Messwerte als Funktion der Zeit animiert dargestellt werden. Dafür bedarf es jedoch externer Programmlogik.

Bei der ersten Ausführung werden initial alle Messwerte der Wasserstände aller Pegelstationen eingelesen. Sobald die Animation durch Klick auf den Start-Button gestartet wurde, werden nacheinander alle Messdaten zu den entsprechenden Zeitpunkten in einer rekursiven Funktion in das Diagramm eingelesen und dargestellt. Dazu muss jedes Mal die aktuelle Datenreihe aus dem Diagramm gelöscht werden um anschließend die neuen Messwerte einlesen und darstellen zu können.
Im Anschluss wird die Position des Sliders, sowie die Darstellung des Zeitpunktes aktualisiert. Dies ist notwendig, weil es sich bei den beiden Elementen um externe, und somit unabhängig vom eigentlichen Diagramm, UI-Elemente handelt.
Nachdem der Zähler der Zeitpunkte hoch gezählt wurde, ruft sich die Funktion, leicht zeitversetzt, wieder rekursiv selbst auf und stellt somit die nächste Datenreihe dar.

Das nachfolgende Struktogramm stellt die funktionsweise des Algorithmus noch einmal vereinfacht dar.

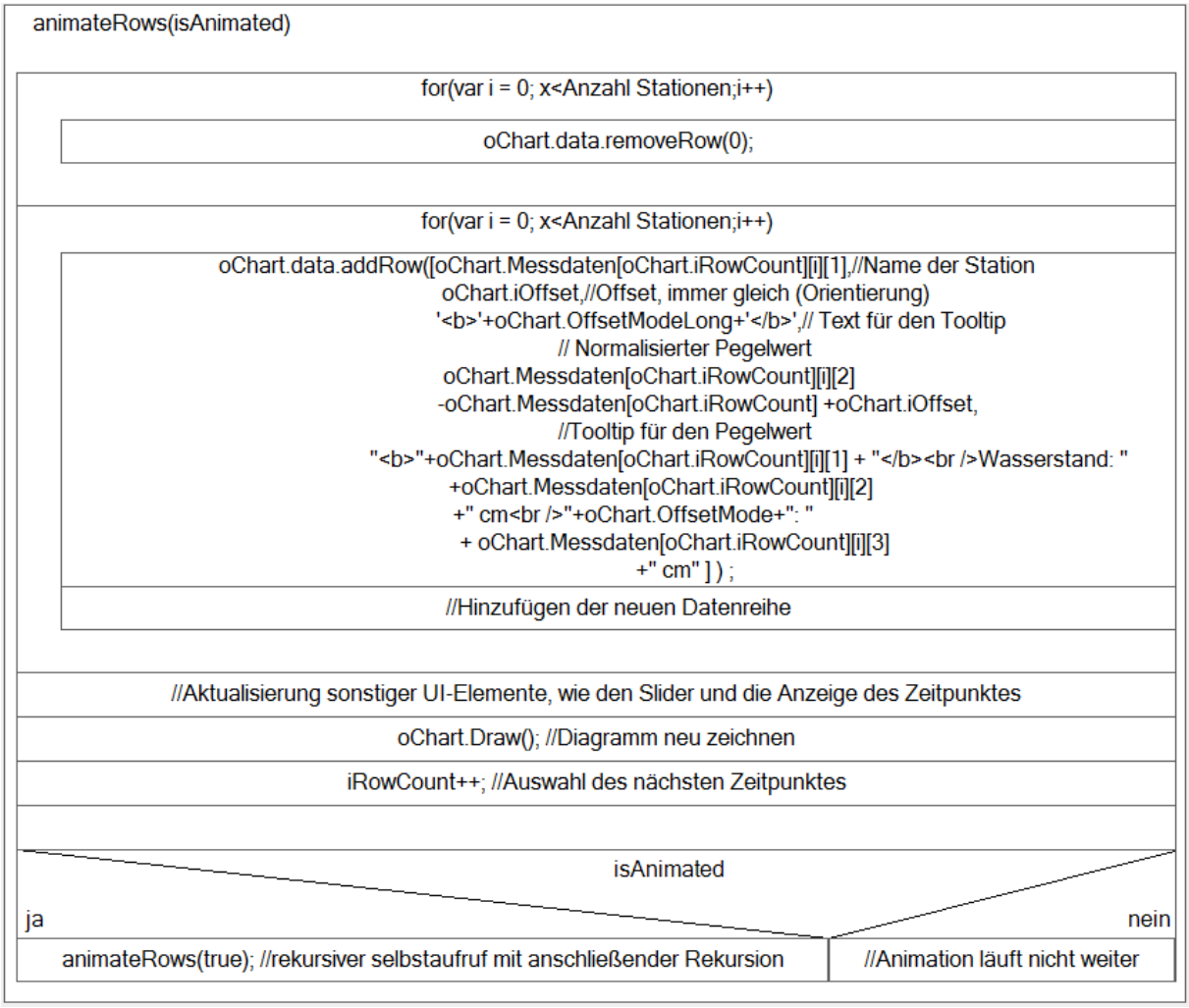

Abbildung 4.3 Struktogramm animateRows

In der Praxis ist diese Funktion etwas komplexer, da sie zusätzlich noch dazu dient, das Diagramm mit neuen Daten darzustellen, wenn der Nutzer mit dem Slider einen spezifischen Zeitpunkt ausgewählt hat.

4.4 Parametrisierung der Anwendung

Die Anwendung wurde unter dem Aspekt entwickelt später in Webseiten eingebunden werden zu können.
Durch die Option einzelne Parameter frei anpassen zu können, ist es möglich eine Anpassung an die Gegebenheiten der umschließenden Webseite umzusetzen. Neben den optischen Modifikation, wie Farben, Größe und Textstil können auch inhaltliche Adaptionen durch Variation der Parameter erfolgen.
So ist es beispielsweise möglich, zu bestimmen in welchem zeitlichen Abstand die Werte dargestellt werden sollen, sodass auch eine Anzeige von je einem Messwert pro Stunde möglich ist.
Realisiert wurde das durch globale Objektvariablen, also Variablen, die von jeder Funktion des Haupt-Javascript-Objektes verwendet werden können.
Durch die Wertzuweisung vor der Initialisierung der Anwendung kann somit die Konfiguration der Anwendung dem Verwendungszweck angepasst werden.
Zusätzlich wertet die Anwendung URL-Parameter aus, mit dessen Hilfe dem Nutzer eine sinnvolle Vorbelegung der Bezugsgröße und dem darzustellenden Zeitintervall vorgegeben wird. Wird die Anwendung beispielsweise mit den URL Parametern „timespan=13&reference=MW“ aufgerufen, so werden die Wasserstände der letzten 14 Tage, bezogen auf den mittleren Wasserstand, dargestellt.
Dies stellt jedoch nur eine Voreinstellung dar und kann vom Nutzer noch geändert werden.
Nachdem die technische Umsetzung besprochen wurde, folgen in den letzten Abschnitten ein Ausblick und eine Zusammenfassung der Arbeit.

5 Ausblick

5.1 Probleme und Verbesserungspotential der Anwendung

Die Anwendung zur Darstellung des Verlaufs der Wasserstände des Rheins über einem spezifischen Zeitpunkt ist grundsätzlich funktionstüchtig und bereit zur Einbindung in eine Webseite.
Dennoch gibt es einige Punkte, die in Zukunft noch verbessert werden könnten.

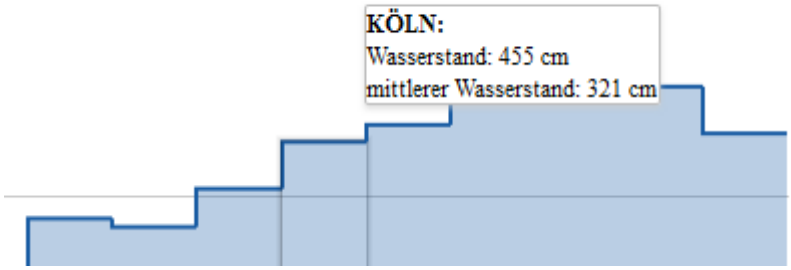

Abbildung 5.1 Darstellung Tooltip

Zeigt man mit dem Mauszeiger auf einen diskreten Wasserstand, so werden detaillierte Informationen als Tooltip angezeigt. Leider unterstützt die Google-Charts-API diese Funktion nur, wenn sich das Diagramm in Ruhe befindet. Somit ist es nicht möglich während der Animation dieses Tooltip zu verwenden. Ein Mittel diesen Misstand zu lösen, wäre die Animation für den Zeitraum der Anzeige des Tooltips anzuhalten, was die Useability jedoch negativ beeinflussen würde.

Ein weiterer Kritikpunkt ist, dass die Performance der Anwendung abhängig von der Leistungsfähigkeit des Clients und des verwendeten Browsers ist. Das ist ein konzeptbedingtes Problem, da Javascript Anwendungen immer von der Javascript-Engine des lokalen Browsers ausgeführt werden. Zum Tragen kommt das Problem vor allem bei dem Internet Explorer bis Version 8.0, in dem die Animation langsamer und weniger flüssig läuft als in anderen Browsern. Da der Internet Explorer 8 schon fast 5 Jahre alt ist und mittlerweile bereits der Internet Explorer 11 verfügbar ist, wird dieses Problem nicht mehr betrachtet.

Ein wünschenswertes Feature für die Zukunft wäre die Möglichkeit auch die Wasserstände anderer Flüsse darstellen zu können. Das ist zwar derzeit theoretisch schon möglich, dazu müssten jedoch noch weitere Parameter angepasst werden, was in diesen Ausführungen nicht näher betrachtet wurde.

5.2 Integration in eine Webseite

Zu Testzwecken wurde die Javascript-Anwendung in eine einfach strukturierte HTML-Seite ohne weiteren Inhalt integriert. Da die Zielsetzung aber in der Einbindung in eine produktive Webseite besteht, muss sich die Anwendung sowohl optisch, als auch technisch einfach integrieren lassen.
Dies wird durch die Verwendung von externen Code und *Stylesheets* ermöglicht. Mit verhältnismäßig wenig Code lässt sich das Gesamtobjekt als ein skalierbares *HTML-Div* einbinden. Zusätzlich müssen dazugehörigen Javascript-Code-Dateien sowie die Google und JQuery-API eingebunden werden. Das erfolgt durch Einfügen des folgenden HTML-Codes in den Head-Tag der Seite:

```
<link rel="stylesheet" href="http://code.jquery.com/ui/1.10.3/themes/smoothness/jquery-ui.css"/>
<link href="style/style.css" rel="stylesheet" type="text/css"/>

<script type="text/javascript" src="https://www.google.com/jsapi"></script>
<script src="http://code.jquery.com/jquery-1.9.1.js"></script>
<script src="http://code.jquery.com/ui/1.10.3/jquery-ui.js"></script>

<script type="text/javascript" src="js/POJlib.js"></script>
<script type="text/javascript" src="js/Pagelib.js"></script>
```

Abbildung 5.2 Inkludieren benötigter Bibliotheken

Zum Ausführen der Anwendung müssen lediglich die Google-Charts-API, sowie die UI-Elemente initialisiert werden, was sich auch auf wenige Zeilen Javascript Code beschränkt.
Die Optik lässt sich durch Anpassen der Stylesheet-Datei nahezu uneingeschränkt modifizieren und somit auch visuell an das Layout der Internetseite anpassen.

6 Zusammenfassung

Im Verlauf der Projektarbeit ist nach der Evaluierung geeigneter Möglichkeiten und Techniken zur dynamischen Darstellung von Pegel-Messdaten eine Anwendung entstanden, die dem Nutzer die Möglichkeit zur Darstellung des Verlaufs der Wasserstände des Rheins in zeitlich animierter Form bietet. Die Wasserstände werden dabei im Querschnitt am Verlauf des Rheins dargestellt. Der Betrachter kann dabei selber entscheiden, in welchem Zeitraum und auf welchen Vergleichswert bezogen die Daten dargestellt werden sollen.
Zusätzlich dazu können sinnvolle Werte bei der Integration in eine Webseite oder der Weitergabe per URL vorgegeben werden.
Da die Messwerte bei jedem Programmstart direkt vom PEGELONLINE REST-Webservice bezogen werden, ist die Darstellung immer aktuell, was im Hochwasserfall für viele Betrachter von großem Nutzen ist.

Vor einer produktiven Integration der Anwendung in eine Webseite, werden noch einige optische und technische Anpassungen im Detail notwendig sein, die grundlegende Funktion ist jedoch schon jetzt gewährleistet.

Literaturverzeichnis

PO01…… http://pegelonline.wsv.de/gast/karte/standard , 06.01.2014

PO02…… http://pegelonline.wsv.de/webservices/zeitreihe/visualisierung?parameter=Wasserstand%20Rohdaten&pegelnummer=23700200 , 06.01.2014

PO03…… http://pegelonline.wsv.de/webservices/rest-api/v2/stations/BONN/W/currentmeasurement.json , 07.01.2014

PO04…… http://pegelonline.wsv.de/webservices/rest-api/v2/stations/BONN/W/measurements.png , 07.01.2014

PO05…… http://pegelonline.wsv.de/webservice/dokuRestapi

OFC01……. http://teethgrinder.co.uk/open-flash-chart/

OFC02……. http://teethgrinder.co.uk/open-flash-chart-2/bar-filled-chart.php, 08.01.2014

FC01…… http://www.flotcharts.org/

JQ01…… http://www.jquery.com/

GC01…….. https://developers.google.com/chart/

HDV01……... http://www.heise.de/developer/artikel/Silverlight-787513.html , 24.01.2014

Glossar

Adobe Flash……..	Plattform zur Programmierung von multimedialen Inhalten. Flash benötigt zur Ausführung im Browser ein entsprechendes Plugin
HTML-Div……..	Container zum Gruppieren und Einbinden von Inhalten; Kann weiteren HTML- und JavaScript-Inhalt enthalten
JavaScript…….	Skriptsprache zur Auswertung von Benutzerinteraktionen, Änderung von Inhalten und Berechnung von Daten in HMTL-Seiten. Jeder moderne Broweser unterstützt die Ausführung von JavaScript-Code
JQuery……..	freie Javascript-Bibliothek mit vielfältigen Funktionsumfang
JSON…….	Datenformat zum Datenaustausch zwischen Anwendungen und Serverdiensten
Parsen…….	zerlegen, auswerten und weiterverarbeiten von beliebigen Daten zur weiteren Verwendung in einer Anwendung
PNG-Format…….	verbreitetes Grafikformat zur Darstellung von Rastergrafiken
Stylesheet…….	engl. Formatvorlage
Tooltip……..	Pop-Up-Fenster zur Darstellung zusätzlicher Informationen beim Verweilen des Mauszeigers über einem Element
Webservice……..	Softwareanwendung zur Bereitstellung von Diensten und Daten

Ehrenwörtliche Erklärung

Ich erkläre ehrenwörtlich,

1. dass ich meine Projektarbeit mit dem Thema:

„Dynamische Visualisierung von Messwerten aus PEGELONLINE zur Präsentation in digitalen Medien“

ohne fremde Hilfe angefertigt habe,

2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und

3. dass ich meine Projektarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

_____ _____

(Ort, Datum) (Unterschrift)

Anlage A Quelltext der Anwendung

Inhalt:

- /Pegelverlauf.html
- /js/POJlib.js
- /js/Pagelib.js

```
 1: <html>
 2:   <head>
 3:
 4:     <link rel="stylesheet"
   href="http://code.jquery.com/ui/1.10.3/themes/smoothness/jquery-ui.css"/>
 5:     <link href="style/style.css" rel="stylesheet" type="text/css"/>
 6:
 7:     <script type="text/javascript"
   src="https://www.google.com/jsapi"></script>
 8:     <script src="http://code.jquery.com/jquery-1.9.1.js"></script>
 9:     <script src="http://code.jquery.com/ui/1.10.3/jquery-ui.js"></script>
10:
11:     <script type="text/javascript" src="js/POJlib.js"></script>
12:     <script type="text/javascript" src="js/Pagelib.js"></script>
13:
14:   </head>
15:
16:   <body >
17:     <div style="margin-left: 20px;height: 700px; ;width: 80%;">
18:         <span id="TitleText"></span>
19:         <div id="TitleIndex">Datenquelle: pegelonline.wsv.de</div>
20:         <div id="TitleIndex2">Bei den verwendeten Daten handelt es sich um
   ungepr&uuml;fte Rohdaten</div>
21:         <div id="chart_div" ></div>
22:         <div id="Legend">
23:             <img src="img/LineRed.png" class="LegendImage" />
24:             <span id="CharacteristicValueText">Mittlerer Wasserstand</span>
25:             <img src="img/LineBlue.png" class="LegendImage" />
26:             gemessener Wasserstand
27:         </div>
28:         <div id="slider" > </div>
29:             <div id="ctrlBTNs"   >
30:                 <a class="ctrlButton" id="b1"  href="#"  >Start</a>
31:                 <a class="ctrlButton" id="b2"  href="#" >Stop</a>

32:             </div>
33:
34:         <span id="table_div"></span>
35:         <div id="TimeSpanDiv">
36:             Darstellung des Pegelverlaufs der letzten
37:             <select id="sTimeSpan" onchange="setNewParameter();" ></select>
38:             Tag(e)
39:         </div>
40:         <div id="Reference">
41:             Darstellung der Wassserst&auml;nde bezogen auf <select
   id="sReference" onchange="setNewParameter();">
42:             </select>
43:
44:         </div>
45:     </div>
46:   </body>
47:       <script language="javascript" type="text/javascript">
48:         //Parameter setzen
49:         google.load('visualization', '1.0', {'packages':['corechart']});
50:         google.load('visualization', '1', {packages:['table']});
51:
52:         //Oberfläche Initialisieren
53:         loadTimespanListBox("sTimeSpan");
54:         loadReferenceListbox("sReference");
```

```
55:         setTimeSpan("sTimeSpan");
56:         setReference("sReference");
57:
58:         //Initialisieren der Diagramm-Optionen
59:         oChart.LoadOptions();
60:
61:         // Diagramm Anzeige Starten
62:         google.setOnLoadCallback(oChart.init);
63:
64:     </script>
65: </html>
```

```
 1:
 2:
 3: //POJlib.js
 4: //Stellt das JS-Objekt zur verfügung,
 5: //welches die Daten hält und den Programmcode ausführt
 6: // Enthält jegliche Logik zur Darstellung und Ausführung der Anwendung
 7:
 8: var oChart = new Object();
 9:
10:
11: oChart.LoadOptions = function()
12: {
13:     ////////////////////////////////////////////////////////
14:     ///Member/Default-Parameter setzen
15:
16:     //JS-ID des Divs, in dem das chart dargestellt wird
17:     oChart.ChartDivID = 'chart_div';
18:     oChart.TableDivID = 'table_div';
19:     oChart.TitleID    =  'TitleText';
20:     oChart.btnStartID = 'b1';
21:     oChart.btnStopID  = 'b2';
22:     oChart.CharacteristicValueLegendID  = 'CharacteristicValueText';
23:
24:     oChart.TitleColor = '#66747F';
25:     oChart.TextColor =  '#6C6F68';
26:
27:
28:
29:     //Zeitperiode aus der die Daten geholt und Dargestellt werden
30:     //sollen
31:     if(oChart.TimeSpanInDays == null || !(oChart.TimeSpanInDays % 1 === 0))
32:     {//Nicht gesetzt, oder kein Integer
33:       oChart.TimeSpanInDays = 14;
34:     }
35:     else
36:     {//Daten dürfen nicht länger als 30 Tage zurückliegen
37:         if(oChart.TimeSpanInDays > 30)
38:             oChart.TimeSpanInDays = 30;
39:     }
40:
41:
42:
43:     //Berechnen der Datumsangaben im Title
44:
45:     oChart.EndDate = new Date();
46:     oChart.EndDate = oChart.EndDate.getDate() +"."
47:                     + oChart.LeadingZero(oChart.EndDate.getMonth()+1) +"."
48:                     +oChart.EndDate.getFullYear();
49:
50:      oChart.StartDate = new Date();
51:      oChart.StartDate.setDate(oChart.StartDate.getDate() -
    oChart.TimeSpanInDays )
52:
53:      oChart.StartDate = oChart.StartDate.getDate() +"."
54:                     +oChart.LeadingZero(oChart.StartDate.getMonth()+1) +"."
55:                     +oChart.StartDate.getFullYear();
56:
57:
58:     document.getElementById(oChart.TitleID ).innerHTML = "Verlauf der
    Wasserstände des Rheins vom " + oChart.StartDate + " bis " +
    oChart.EndDate;
```

```
59:
60:     //Diagramm-Optionen
61:     oChart.options = {
62:         //Globale Einstellungen
63:         fontName: 'Arial',
64:
65:         titleTextStyle :{
66:             color: oChart.TitleColor,
67:             fontSize : 27
68:         },
69:
70:         //Title wird in einem externen HTML-Div verwaltet
71:         titlePosition : 'none',
72:
73:
74:         //HTML für Tooltip aktivieren --> Überschrift FETT
75:         tooltip: {isHtml: true},
76:
77:         //Animation beim auswechseln der Daten
78:         animation:{
79:             duration: 200,
80:             easing: 'linear'
81:         },
82:
83:         //Größe des eigentlichen Diagramms im DIV
84:         chartArea:{
85:             width : "100%",
86:             height : "100%",
87:             top : "10%",
88:             left :"0%"
89:
90:
91:         },
92:
93:         //Formatieren der Datenreihen:
94:             //Charakteristische Daten  als Offset --> Rote Linie
95:             //Eigentliche Messwerte als verbundene Balken in Blau
96:         series: [{color :'red',areaOpacity : 0},{color:'#195ca3'}],
97:
98:         //Formatieren der Y-Achse
99:         vAxis:{
100:            textPosition : 'none',
101:            viewWindow:{
102:             viewWindowMode:'explicit',
103:             //Werte für Minimum und Maximum vorerst nicht von bedeutung
104:             //Werden später durch den Code Dynamisch angepasst anhand der
105:             //Messwerte
106:             min :  0,
107:             max : 100
108:           }
109:         },
110:
111:         //Formatieren der X-Achse
112:         hAxis:
113:         {
114:             textPosition: 'in',
115:             allowContainerBoundaryTextCufoff :'true',
116:             //Jede 2. Beschriftung wird angezeigt
117:             showTextEvery:2,
```

```
118:             textStyle:{
119:                 color: oChart.TextColor,
120:                 fontSize:14
121:             }
122:
123:
124:         }
125:
126:     };
127:
128:     //Zähler für das Umschalten der Messdaten
129:     oChart.iRowCount = 0;
130:
131:     //Stop-Flag zum unterbrechen der Animation
132:     oChart.bStopAni = false;
133:
134:     //Läuft die Animation akutell?
135:     oChart.bAnimationRunning = false;
136:
137:     //Woran soll sich die Darstellung Orientieren?
138:     if(oChart.OffsetMode == null)
139:     {
140:         oChart.OffsetMode ="MW";
141:     }
142:
143:
144:     //Beschriftung einfügen
145:     if(oChart.OffsetModeLong == null)
146:     {
147:       oChart.OffsetModeLong = "Mittleres Hochwasser";
148:     }
149:
150:     //extern erstellte Legende beschriften
151:     document.getElementById(oChart.CharacteristicValueLegendID ).innerHTML
    = oChart.OffsetModeLong;
152:
153:
154:     //Welches Gewässer soll dargestellt werden?
155:     oChart.Water = "Rhein";
156:     //Pegel Darstellungs-Offset
157:     //Der  Wert wird nach der Datenübernahme berechnet, sodass kein Pegel
158:     //unterhalb der X-Achse gelangen kann
159:     oChart.iOffset = 0;
160:
161:
162:     //Deklarieren der Buttons um sicher gehen das sie
163:     //vorhanden sind.
164:     oChart.button_start = document.getElementById(oChart.btnStartID);
165:     oChart.button_stop = document.getElementById(oChart.btnStopID);
166:
167:     //Deklarieren der Chart-Daten und Darstellung um sicher zu gehen,
168:     //das Sie vorhanden sind
169:     oChart.data = null;
170:     oChart.chart = null;
171:
172:
173:     //Deklarieren der Messdaten um sicher zu gehen,
174:     //das Sie vorhanden sind
175:     oChart.Messdaten = null;
```

```
176:
177: }
178:
179:
180:
181:
182:
183: ////////////////////////////////////////////////////////////
184: //Methoden
185:
186:
187: oChart.init = function()
188: {//Initialisiert die Darstellung des Diagramms
189: //Jegliche Veränderung an den Parametern muss vor der ersten
190: //Ausführung dieser Funktion erfolgen
191:
192:
193:     // Datentabelle
194:     oChart.data = new google.visualization.DataTable();
195:
196:     // Erstellen des Visualierungsobjektes
197:     oChart.chart = new
    google.visualization.SteppedAreaChart(document.getElementById(oChart.ChartDivID));
198:
199:     //Initialisieren der Buttons/Übergabe in den Code
200:     oChart.button_start = document.getElementById(oChart.btnStartID);
201:     oChart.button_stop = document.getElementById(oChart.btnStopID);
202:
203:
204:     //Messdaten von Pegel Online einlesen
205:     oChart.Messdaten = oChart.fillData();
206:
207:     //Maximumwert der Y-Achse setzen
208:     oChart.SetMaxAxisValues()
209:
210:     //Tabelle vorbereiten/Tabellenkopf
211:         //Messstelle
212:     oChart.data.addColumn('string','Messtelle');
213:         //Welche Characteristische Werte dienen als Orientierung
214:     oChart.data.addColumn('number',oChart.OffsetModeLong);
215:         //Tooltip für den Characteristischen Wert
216:     oChart.data.addColumn({type: 'string', role: 'tooltip', 'p': {'html':
    true}});
217:         //Pegelwert
218:     oChart.data.addColumn('number','gemessener Wassserstand');
219:         //Tooltip für den Pegelwert
220:     oChart.data.addColumn({type: 'string', role: 'tooltip', 'p': {'html':
    true}});
221:
222:
223:     //Daten initial in das Diagramm laden
224:     for(var i = 0; i<oChart.Messdaten[0].length;i++)
225:     {//Für jede verfügbare Station
226:         //Hinzufügen der Zeile zu dem DatenArray
227:         oChart.data.addRow([oChart.Messdaten[0][i][1],//Name der Station
228:                             oChart.iOffset,//Offset, immer gleich
    (Orientierung)
229:                             '<b>'+oChart.OffsetModeLong+'</b>', // Text
    für den Tooltip
```

```
230:                                // Normalisierter Pegelwert
231:                                oChart.Messdaten[0][i][2]-
232:                                oChart.Messdaten[0][i][3]+
233:                                oChart.iOffset,//
234:                                //Tooltip für den Pegelwert
235:                                "<b>"+oChart.Messdaten[0][i][1]+ //Name der
    Messstelle
236:                                ":</b><br />Wasserstand:
    "+oChart.Messdaten[0][i][2]+ //Pegelwert
237:                                " cm<br />"+oChart.OffsetModeLong+": " + //
    Name der Characteristischen Werte
238:                                oChart.Messdaten[0][i][3]+" cm"] ) ;//Wert
239:     }
240:
241:
242:     var oDate = oChart.toDate(oChart.Messdaten[0][0][0]);
243:     //Aktualisieren des Zeitpunktes:
244:     document.getElementById(oChart.TableDivID).innerHTML =
    oChart.LeadingZero(oDate.getDate())+"."+oChart.LeadingZero(oDate.getMonth()+1)+"."+
245:     "  " + oChart.LeadingZero(oDate.getHours()) +":"+
    oChart.LeadingZero(oDate.getMinutes()) ;
246:
247:     //Eventhandler für die onclick-Methoden der Buttons

248:     oChart.button_start.onclick = oChart.StartAnimation;
249:     oChart.button_stop.onclick = oChart.StopAnimation;
250:
251:     //Neuzeichnen des Diagramms beim verändern der Bildschirmgröße
252:     //und des Zoom-Levels
253:     window.onresize = function() {
254:         oChart.drawChart();
255:     }
256:
257:     //Erstellen und Verwalten des Timeline-Sliders
258:     oChart.handleSlider();
259:
260:
261:     //Initiales Zeichnen des Graphen
262:     oChart.drawChart();
263:
264:
265:
266: }
267:
268:
269: ////////////////////////////////////////////////////////////////////
270: /////getJSON()
271: ///Funktion: Löscht die die Aktuelle Datenzeile (Je Zeiteinheit)
272: //und fügt die nächste ein. Dies geschieht in einer Endlosschleife
273: ///Parameter:
274: //(String) searchstring
275: ///Rückgabewert:
276: //Ein geparstes JSON Objekt
277: oChart.getJSON = function(searchstring)
278: {
279:     var url = "http://www.pegelonline.wsv.de/webservices/rest-api/v2/"+
    searchstring ;
280:     var xmlHttp = null;
281:     var jsonobj = null;
```

```
282:
283:     xmlHttp = new XMLHttpRequest();
284:     xmlHttp.open( "GET", url, false );
285:     xmlHttp.send( null );
286:
287:     //Der JSON String wird in ein Objekt geparsed
288:     jsonobj = eval(xmlHttp.responseText);
289:
290:     return jsonobj;
291: }
292:
293: ////////////////////////////////////////////////////////////////////
294: /////AnimateRows()
295: ///Funktion: Löscht die die Aktuelle Datenzeile (Je Zeiteinheit)
296: //und fügt die nächste ein. Dies geschieht rekursiv in einer Endlos
297: //Schleife und kann durch setzen eines Flags unterbrochen werden
298: ///Parameter:
299: //bAnimation - bestimmt ob die Funktion rekursiv ausgeführt wird
300:
301: oChart.AnimateRows = function(bAnimation){
302:     //Reihen aus Datenarry löschen Löschen
303:     for(var x = 0; x<oChart.Messdaten[0].length;x++)
304:     {
305:        oChart.data.removeRow(0);
306:     }
307:
308:     //Reihen Hinzufügen
309:     for(var i = 0; i<oChart.Messdaten[0].length;i++)
310:     {
311:        oChart.data.addRow([oChart.Messdaten[oChart.iRowCount][i][1],//Name
     der Station
312:                            oChart.iOffset,//Offset, immer gleich
     (Orientierung)
313:                            '<b>'+oChart.OffsetModeLong+'</b>',// Text für den
     Tooltip
314:                            // Normalisierter Pegelwert
315:                            oChart.Messdaten[oChart.iRowCount][i][2]
316:                            -oChart.Messdaten[oChart.iRowCount][i][3]
317:                            +oChart.iOffset,
318:                            //Tooltip für den Pegelwert
319:                            "<b>"+oChart.Messdaten[oChart.iRowCount][i][1]
320:                            +"</b><br />Wasserstand: "
321:                            +oChart.Messdaten[oChart.iRowCount][i][2]
322:                            +" cm<br />"+oChart.OffsetMode+": "
323:                            + oChart.Messdaten[oChart.iRowCount][i][3]
324:                            +" cm" ] ) ;
325:     }
326:
327:     //UTC-String in Datum konvertieren
328:     var oDate = oChart.toDate(oChart.Messdaten[oChart.iRowCount][0][0]);
329:
330:
331:     //Aktualisieren des Zeitpunktes:
332:     document.getElementById(oChart.TableDivID).innerHTML =
     oChart.LeadingZero(oDate.getDate())+"."+oChart.LeadingZero(oDate.getMonth()+1)+"."+
333:     "  " + oChart.LeadingZero(oDate.getHours()) +":"+
     oChart.LeadingZero(oDate.getMinutes()) ;
334:     //Neuzeichnen des Diagramms
335:     oChart.drawChart();
```

```
336:
337:
338:     if(bAnimation)
339:     {//Rekursive Animation erwünscht
340:         oChart.bAnimationRunning = true;
341:         //Selbstaufruf mit leichter verzögerung
342:         window.setTimeout( oChart.Play, 100);
343:     }
344:     else
345:     {
346:         oChart.bAnimationRunning = false;
347:     }
348:
349: }
350: ////////////////////////////////////////////////////////////////////
351: /////StartAnimation()
352: ///Funktion:Initiiert die Animation
353: ///Parameter:
354: //Keine
355:
356: oChart.StartAnimation = function()
357: {
358:     if(oChart.button_start.innerHTML == "Start")
359:     {
360:         oChart.button_start.innerHTML = "Pause";
361:         oChart.iRowCount =  $( "#slider" ).slider( "value" );
362:         oChart.bStopAni = false;
363:         oChart.Play();
364:     }
365:     else
366:     {
367:        oChart.PauseAnimation();
368:     }
369:
370: }
371:
372: ////////////////////////////////////////////////////////////////////
373: /////StopAnimation()
374: ///Funktion:Startetund unterbricht die Ausführung der Animation
375: //indirekt recursiver Aufruf durch AnimateRows()
376: ///Parameter:
377: //Keine
378: oChart.Play = function()
379: {
380:
381:
382:     if(oChart.bStopAni == false)
383:     {
384:
385:         oChart.iRowCount = oChart.iRowCount + 1;
386:
387:         //Position des Sliders updaten
388:         $( "#slider" ).slider( "option", "value",oChart.iRowCount  );
389:
390:         if(oChart.iRowCount >= oChart.Messdaten.length - 4  )
391:         {
392:             oChart.iRowCount =0;
393:             oChart.StopAnimation();
394:         }
```

```
395:         else
396:         {
397:          oChart.AnimateRows(true);
398:         }
399:
400:     }
401:     else
402:     {
403:       oChart.bStopAni = false;
404:       //Optisches Zurücksetzen des Diagramms;
405:       oChart.AnimateRows(false);
406:     }
407:
408: }
409:
410: ////////////////////////////////////////////////////////////////////
411: /////NextDataSet()
412: ///Funktion:Ruft nächsten Datensatz ab und pausiert die animaiton
413: ///Parameter:
414: //Keine
415: oChart.NextDataSet = function()
416: {
417:     if(oChart.bAnimationRunning)
418:     {
419:          oChart.PauseAnimation();
420:     }
421:     oChart.iRowCount = oChart.iRowCount + 1 ;
422:
423:     if(oChart.iRowCount == oChart.Messdaten.length - 4  )
424:     {
425:         oChart.iRowCount =0;
426:     }
427:
428:     oChart.AnimateRows(false);
429:     //Position des Sliders updaten
430:     $( "#slider" ).slider( "option", "value",oChart.iRowCount  );
431: }
432:
433: ////////////////////////////////////////////////////////////////////
434: /////PrevDataSet()
435: ///Funktion:Ruft vorherigen Datensatz ab und pausiert die animaiton
436: ///Parameter:
437: //Keine
438: oChart.PrevDataSet = function()
439: {
440:     if(oChart.bAnimationRunning)
441:     {
442:          oChart.PauseAnimation();
443:     }
444:
445:     oChart.iRowCount = oChart.iRowCount - 1;
446:
447:     if(oChart.iRowCount == -1  )
448:     {
449:         oChart.iRowCount =oChart.Messdaten.length -4;
450:     }
451:
452:     oChart.AnimateRows(false);
453:     //Position des Sliders updaten
```

```
454:     $( "#slider" ).slider( "option", "value",oChart.iRowCount  );
455: }
456:
457:
458: ////////////////////////////////////////////////////////////////////
459: /////StopAnimation()
460: ///Funktion:Setzt Flags und beendet somit die Ausführung der Animation
461: // Und setzt den Zählerstand auf 0
462: ///Parameter:
463: //Keine
464: oChart.StopAnimation = function()
465: {
466:
467:
468:     oChart.iRowCount = 0;
469:     oChart.bStopAni = true;
470:
471:     //Position des Sliders updaten
472:     $( "#slider" ).slider( "option", "value",oChart.iRowCount  );
473:     oChart.button_start.innerHTML = "Start";
474:     oChart.AnimateRows(false);
475:
476: }
477:
478: ////////////////////////////////////////////////////////////////////
479: /////PauseAnimation()
480: ///Funktion:Setzt Flags und beendet somit die Ausführung der Animation
481: ///Parameter:
482: //Keine
483: oChart.PauseAnimation = function()
484: {
485:     oChart.bStopAni = true;
486:     oChart.button_start.innerHTML = "Start";
487:
488: }
489:
490: ////////////////////////////////////////////////////////////////////
491: /////FillData()
492: ///Funktion:Holt die Messdaten von PEGEL ONLINE per REST-API
493: //Erzeugt Array mit Daten und gibt dieses zurück
494: ///Parameter:
495: //Keine
496: ///Rückgabewert:
497: //Array mit Mess Daten
498: oChart.fillData = function()
499: {
500:     //Daten per JSON von PEGEL ONINE holen
501:
502:     //Die verfügbaren Stationen holen
503:     var oStations = oChart.getJSON("stations.json?waters=" +
     oChart.Water+"&includeTimeseries=true&includeCharacteristicValues=true");
504:     var MeasureData = new Array();
505:     var lclcounter =  0;
506:     var MinVal = 0;
507:
508:
509:     //Wieviele Stationen mit hinterlegten verwendeten! Characteristischen
     eigentschaften gibt es?
510:     for(var z = 0; z<oStations.length;z++)
```

```
511:     {
512:        if(oStations[z].timeseries[0].characteristicValues.length >0)
513:        {
514:            for(var u = 0; u <
    oStations[z].timeseries[0].characteristicValues.length;u++)
515:            {
516:
    if(oStations[z].timeseries[0].characteristicValues[u].shortname ==
    oChart.OffsetMode)
517:                {
518:
519:                  lclcounter++;
520:                }
521:            }
522:        }
523:
524:     }
525:
526:     //Das Array Initialisieren
527:     for(var i = 0; i<(oChart.TimeSpanInDays*12);i++)
528:     {//Für jede Uhrzeit = Zeile
529:         MeasureData[i] = new Array();
530:         for(var z = 0; z<lclcounter;z++)
531:         {//Jede Station
532:             MeasureData[i][z] = new Array();
533:
534:
535:             MeasureData[i][z][2] = 0;
536:
537:         }
538:     }
539:
540:     //Befüllen der Daten
541:     lclcounter = 0;
542:     for(var z = 0; z<oStations.length;z++)
543:     {
544:        if(oStations[z].timeseries[0].characteristicValues.length >0 )
545:        {
546:            var iOrientation = null;
547:            for(var u = 0; u <
    oStations[z].timeseries[0].characteristicValues.length;u++)
548:            {
549:
    if(oStations[z].timeseries[0].characteristicValues[u].shortname ==
    oChart.OffsetMode)
550:                {
551:                  iOrientation =
    oStations[z].timeseries[0].characteristicValues[u].value;
552:                }
553:            }
554:            if(iOrientation > 0)
555:            {
556:                oTempMeasurements = oChart.getJSON("stations/"+
    oStations[z].uuid
    +"/W/measurements.json?start=P"+oChart.TimeSpanInDays+"D");
557:               for(var i= 0 ; i< oTempMeasurements.length;i += 8)
558:               {
559:                    //Ermitteln des Minimalen Wertes, zur Offsetermittlung
560:                    if((oTempMeasurements[i].value - iOrientation) <
    MinVal )
```

```
561:                     {
562:                       MinVal = oTempMeasurements[i].value - iOrientation;

563:                     }
564:
565:                     MeasureData[i/8][lclcounter][0] =
    oTempMeasurements[i].timestamp; // Zeitpunkt
566:                     MeasureData[i/8][lclcounter][1] =
    oStations[z].longname; //Name der Station
567:                     MeasureData[i/8][lclcounter][2] =
    oTempMeasurements[i].value;// Messwert
568:                     MeasureData[i/8][lclcounter][3] = iOrientation;
    //Bezugswert
569:
570:                 }
571:                 lclcounter ++;
572:             }
573:
574:         }
575:     }
576:
577:     //Festlegen des Offsets, sodas die Pegel niemals unter der Y-Achse
    liegen können
578:
579:     oChart.iOffset = 50+ (MinVal * (-1));
580:      return MeasureData;
581: }
582:
583:
584: ////////////////////////////////////////////////////////////////////
585: /////StopAnimation()
586: ///Funktion:berechnet die Maximale Höhe der X-Achse und setzt den Wert
587: ///Parameter:
588: //Keine
589: ///Rückgabewert:
590: //keine
591: oChart.SetMaxAxisValues = function()
592: {
593:     var tMax = oChart.iOffset;
594:     for(var i = 0;i<oChart.Messdaten.length;i++)
595:     {
596:         for(var z=0;z<oChart.Messdaten[i].length;z++)
597:         {
598:             if((oChart.Messdaten[i][z][2]-oChart.Messdaten[i][z][3]+
599:                                 oChart.iOffset) > tMax)
600:                 tMax = (oChart.Messdaten[i][z][2]-
    oChart.Messdaten[i][z][3]+
601:                                 oChart.iOffset);
602:         }
603:     }
604:
605:     oChart.options.vAxis.viewWindow.max = tMax;
606: }
607:
608: oChart.drawChart = function()
609: {
610:   // Disabling the button while the chart is drawing.
611:
612:   oChart.chart.draw(oChart.data, oChart.options);
```

```
613:
614: }
615:
616: oChart.handleSlider = function()
617: {
618:     //Den Slider verwalten:
619:     //Slider erstellen mit JQuery:
620:     $( "#slider" ).slider({
621:         slide: function( event, ui ) {}
622:     },{ min: 0 },{ max: oChart.TimeSpanInDays*12 });
623:
624:     $( "#slider" ).on( "slide", function( event, ui ) {
625:     //Was geschieht beim Sliden
626:
627:         oChart.PauseAnimation();
628:
629:         oChart.AnimateRows();
630:         oChart.iRowCount =  $( "#slider" ).slider( "value" );
631:     } );
632: }
633:
634: oChart.LeadingZero = function(NumberValue)
635: {
636:     if(NumberValue <10)
637:         return "0"+NumberValue;
638:
639:     return NumberValue;
640:
641: }
642:
643: oChart.toDate = function(DateString)
644: {// Konvertiert einen Datumsstring zu eine Datumsobjekt
645: //  String sollte im UTC-Format sein
646:     DateString =  DateString.split("+")[0];
647:     DateString = DateString.split("T");
648:
649:     var TimeString = DateString[1];
650:     TimeString = TimeString.split(":");
651:     DateString = DateString[0].split("-");
652:
653:     return new Date(parseInt(DateString[0]),parseInt(DateString[1])-1,
    parseInt(DateString[2]),parseInt(TimeString[0]),parseInt(TimeString[1]),
    parseInt(TimeString[2]));
654:
655: }
```

```
1: //Pagelib.js
2:
3: //Stellt Methoden zur Verwaltung der HTML-Seite
4:
5: function getQueryVariable(variable) {
6: //Sucht in den GET-Übergabeparametern nach Variablen und gibt deren
7: //Wert zurück
8:     var query = window.location.search.substring(1);
9:     var vars = query.split("&");
10:    for (var i=0;i<vars.length;i++) {
11:        var pair = vars[i].split("=");
12:        if (pair[0] == variable) {
13:        return pair[1];
14:        }
15:    }
16: }
17:
18: function setNewParameter()
19: { //Ruft die Seite erneut mit den veränderten Parametern auf
20:     location.replace(window.location.pathname +"?timespan="+
    (document.getElementById('sTimeSpan').selectedIndex) +
21:     "&reference=" +
    document.getElementById('sReference').options[document.getElementById('sReference'
22: }
23:
24: function setTimeSpan(ListBoxID)
25: { // Setzt die Auswahl in der Listbox auf den Übergabeparameter
26: //und übergibt den Wert an das Diagramm / POJlib.js
27:     if(getQueryVariable("timespan") % 1 === 0 )
28:     {
29:         if(getQueryVariable("timespan") > 30)
30:         { //Verhindert das zu große Werte übergeben werden
31:            oChart.TimeSpanInDays = 30;
32:            document.getElementById(ListBoxID).selectedIndex = 29;
33:         }
34:         else if(getQueryVariable("timespan") < 1)
35:         {
36:             oChart.TimeSpanInDays = 1;
37:             document.getElementById(ListBoxID).selectedIndex = 0;
38:         }
39:         else
40:         {
41:             oChart.TimeSpanInDays = parseInt(getQueryVariable("timespan"))
    + 1;
42:             document.getElementById(ListBoxID).selectedIndex =
    getQueryVariable("timespan");
43:         }
44:
45:     }
46:     else
47:     {
48:       document.getElementById(ListBoxID).selectedIndex = 13;
49:     }
50:
51: }
52:
53: function setReference(ListBoxID)
54: {     // Setzt die Auswahl des Bezugswertes in der Listbox auf
55: //den Übergabeparameter und übergibt den Wert an das Diagramm / POJlib.js
```

```
56:
57:     switch(getQueryVariable("reference"))
58:     {
59:         case "MHW":
60:                 oChart.OffsetMode = getQueryVariable("reference");
61:                 oChart.OffsetModeLong = "mittleres Hochwasser";
62:             break;
63:         case "MW":
64:                 oChart.OffsetMode = getQueryVariable("reference");
65:                 oChart.OffsetModeLong = "mittlerer Wasserstand";
66:                 CharacteristicValueText
67:              break;
68:         case "MNW":
69:                 oChart.OffsetMode = getQueryVariable("reference");
70:                 oChart.OffsetModeLong = "mittleres Niedrigwasser";
71:              break;
72:         default:
73:                 oChart.OffsetMode = "MW";
74:                 oChart.OffsetModeLong = "mittlerer Wasserstand";
75:
76:     }
77: }
78:
79: function loadTimespanListBox(ListBoxID)
80: {  // Initialisiert die Listbox für mögliche Intervalle
81:     for(var i = 0 ; i < 30 ;i++)
82:     {
83:         document.getElementById(ListBoxID).options[i] = new Option(i+1,i);
84:
85:     }
86: }
87:
88: function loadReferenceListbox(ListBoxID)
89: { // Initialisiert die Listbox für mögliche Bezugswerte
90:
91:    if(getQueryVariable("reference") == "MW")
92:         document.getElementById(ListBoxID).options[0] =  new
    Option("mittleren Wasserstand","MW",false,true);
93:     else
94:         document.getElementById(ListBoxID).options[0] =  new
    Option("mittleren Wasserstand","MW");
95:
96:    if(getQueryVariable("reference") == "MNW")
97:         document.getElementById(ListBoxID).options[1] = new
    Option("mittleres Niedrigwasser","MNW",false,true);
98:     else
99:         document.getElementById(ListBoxID).options[1] = new
    Option("mittleres Niedrigwasser","MNW");
100:
101:     if(getQueryVariable("reference") == "MHW")
102:         document.getElementById(ListBoxID).options[2] = new
    Option("mittleres Hochwasser","MHW",false,true);
103:     else
104:         document.getElementById(ListBoxID).options[2] = new
    Option("mittleres Hochwasser","MHW");
105:
106: }
```