

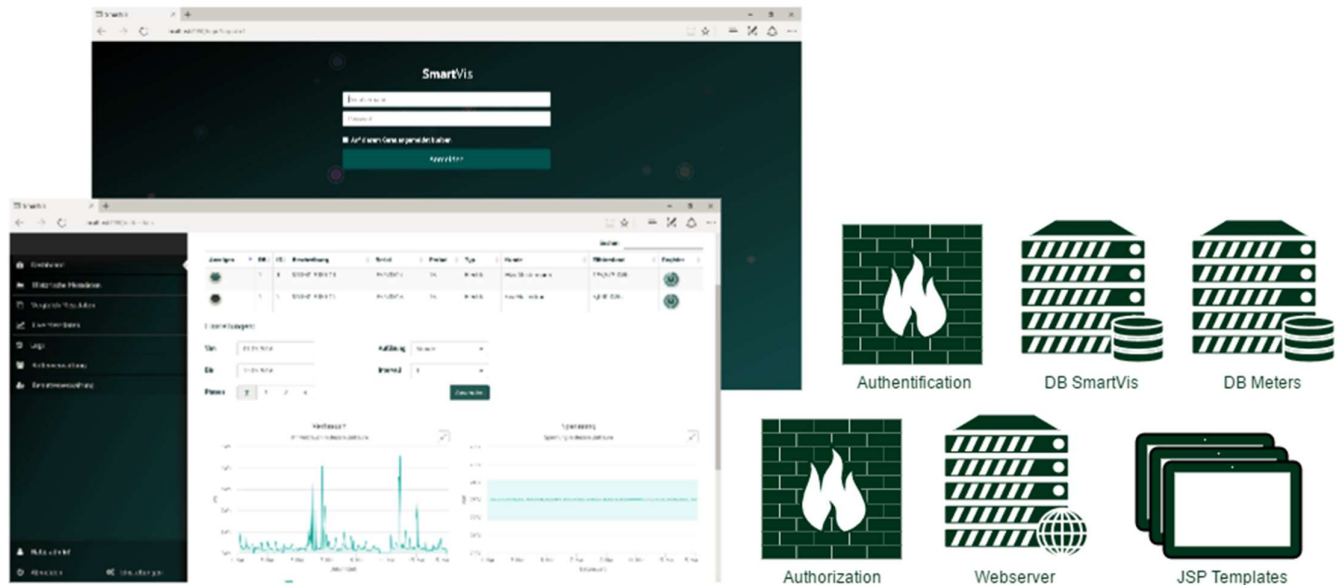
# R. & D. Projekt

## SmartVis-Dokumentation

### Visualisierung von Smart Meter-Daten

(FH-Salzburg / ITS-B-M2015 / Unger, Moser, Wurz)

Stand 18.09.2016



## Inhalt

1	Einleitung .....	3
2	Konzepte und eingesetzte Technologien.....	4
2.1	Verwendetes Architekturmodell .....	4
2.2	Spring MVC Konfiguration .....	4
2.3	Build- und Dependency-Management Maven .....	5
3	Konzeption der Webschnittstelle .....	6
4	Anwendungskomponenten .....	9
4.1	Model.....	9
4.2	View .....	14
4.3	Controller.....	20
5	Sicherheitsaspekte.....	22
5.1	Authentifizierung und Autorisierung (engl. AAA).....	22
5.2	Berechtigungskonzept .....	22
6	Einstellungen SmartVis .....	25
6.1	SmartVis-Anwendungseinstellung.....	25
6.2	Datenbank konfigurieren .....	26
6.3	H0-Profil - Datenbasis .....	27
6.4	Benutzer- und Rollenverwaltung .....	27
7	Installation SmartVis auf Raspberry Pi.....	29
7.1	Voraussetzungen .....	29
7.2	SmartVis-Datenbank anlegen .....	29
7.3	SmartVis-Anwendung auf Raspberry Pi installieren (Deployment).....	29
7.4	SmartVis-Anwendung auf Raspberry Pi automatisch starten.....	29
8	SmartVis Entwicklung.....	31
8.1	Benötigte Komponenten: .....	31
8.2	Starten / Debuggen der Anwendung .....	31

## 1 Einleitung

SmartVis ist eine Software-Anwendung zur Visualisierung von Smart Meter Daten. Die relevanten Smart Meter Messdaten und Metadaten werden in Gateways und Datacenter gehalten.

- Gateways (GW) sind Systeme in der Domäne von Endkunden, die zyklisch Messdaten von angebundenen Smart Meter Geräten abfragen und speichern.
- Datacenter (DC) sind Systeme in der Domäne von Betreiber-Firmen, die zyklisch aggregierte Messdaten von Gateways erhalten und speichern.

Gateways und Datacenter haben grundsätzlich idente Datenmodelle und werden nicht technisch, sondern fachlich (durch entsprechende Berechtigungen) unterschieden.

In der vorgegebenen Smart Meter Infrastruktur ist die Ausführung der Anwendung auf Raspberry Pi Systemen mit Linux Betriebssystem vorgesehen. Die Persistierung von Smart Meter-Daten erfolgt durch ein Vorsystem in einer JDBC-Datenbank und ist nicht Teil der SmartVis-Anwendung. Das physische Datenmodell für die zu visualisierenden Informationen ist somit für die SmartVis-Anwendung vorgegeben.

Die Anzeige von Smart Meter-Daten erfolgt jeweils auf aktuellen, HTML5 / CSS3-fähigen Web-Browsern. Die benötigten Web-Server-Komponenten werden auf den GW- beziehungsweise DC-Systemen abgebildet. Als Entwicklungssprache kommt Java zum Einsatz, der Datenzugriff wird mit JDBC realisiert. Die Systeme sind über IP erreichbar.

Der Inhalt und der Umfang der Web-Seiten zur Datenvisualisierung sind kontextabhängig. Vom Gateway werden eine Kunden- und eine Techniker-Sicht zur Verfügung gestellt, vom DataCenter ist eine Betreiber-Sicht abrufbar. Weiters gibt es eine Show-Sicht zur gemischten Anzeige von DC- und GW-Daten. Die Kontextunterscheidung erfolgt über den authentifizierten Benutzer und dessen Rolle. Damit die Rollen der Anwender unterscheidbar sind, ist eine Benutzer-Authentifizierung erforderlich.

## 2 Konzepte und eingesetzte Technologien

### 2.1 Verwendetes Architekturmodell

Als Architekturmodell wird das MVC-Paradigma herangezogen, da es wesentliche strukturelle Vorteile wie eine aufgabenorientierte Unterteilung der Applikation vorsieht und somit die Modularität und Austauschbarkeit der einzelnen Komponenten fördert. Für die Implementierung des MVC-Paradigmas wird das aktionsbasierte Framework Spring MVC verwendet.

### 2.2 Spring MVC Konfiguration

Um das zentrale Servlet in der Spring MVC Struktur, das sogenannte Dispatcher Servlet, im Webserver zu registrieren wird eine eigene Initialisierungsklasse, kurz angelegt. Zudem werden Konfigurationsklassen für die Webapplikation selbst als auch für den Datenbankzugriff angelegt. Sie benötigen die Annotation *Configuration* um sie gegenüber Spring MVC als Konfigurationsklassen auszuweisen. Diese Konfigurationsklassen werden dann im Initializer beim DispatcherServlet registriert. Somit wird Spring MVC in die Lage versetzt, nachdem der Webserver den ServletContext geladen hat, sich um die weitere Konfiguration der Webapplikation, entsprechend dem Inhalt der Konfigurationsklassen, zu kümmern. Außerdem wurde im Initializer festgelegt, dass das DispatcherServlet für alle Anfragen die erste Anlaufstelle ist.

Spring bietet einige Automatismen an, die in der Java-basierten Konfiguration über spezielle Annotationen aktiviert werden können. In der Konfigurationsklasse für die Webapplikation wird mit der Annotation *EnableWebMvc* die annotationsbasierte Konfiguration der Spring MVC Komponenten aktiviert. Mittels der *ComponentScan* Annotation wird festgelegt, in welchem Package Spring MVC sogenannte Komponenten findet um diese dem Applikationskontext hinzuzufügen.

Eine Komponente ist im Spring-Kontext eine Klasse, die als Controller, Service oder Repository annotiert ist und den drei Schichten des MVC zugeordnet werden kann, also die Präsentationsschicht, die Businesslogik beherbergende Schicht und die Datenschicht. Es gibt aber auch die Annotation als einfache Komponente mittels *Component* um auch Klassen die nicht einer dieser Schichten zugeordnet sind, im Applikationskontext registrieren zu können.

Der große Vorteil einer Komponente besteht darin, dass Spring diese durch die sogenannte Auto-Erkennung mittels des Scannens des Klassenpfades, aktiviert durch die *ComponentScan* Annotation, dem Applikationskontext hinzufügt. Alle dem Applikationskontext hinzugefügten Komponenten stehen unter der Verwaltung von Spring MVC und können durch die Annotation *Autowired* in eine andere Klasse *injiziert* werden. Dieses Prinzip der Bereitstellung von Abhängigkeiten in eine Klasse hinein nennt man *Dependency Injection*. Es fördert die größtmögliche Unabhängigkeit von anderen Java Klassen und steigert dadurch die Möglichkeit der Wiederverwendbarkeit und unabhängigen Testbarkeit von Klassen. Es stellt eines der Kernprinzipien Springs dar.

In den Konfigurationsklassen werden nach dem gleichen Prinzip mit der Annotation *Bean* Objekte im Applikationskontext registriert. So wird die Klasse zum Validieren der, durch einen Nutzer mit Administrationsrechten, modifizierten Attribute des Datenbankzugriffs, auf diese Art und Weise verwaltet. Außerdem wird in der Konfigurationsklasse für die Webapplikation noch eingestellt, in welchem Verzeichnis die Views zu finden sind. Zudem wird Spring MVC die Frontend spezifische Ressourcenverwaltung übertragen, indem deren Ort über die *addResourceHandlers* Methode im *ResourceHandlerRegistry* von Spring registriert wird. Diese Methode wird durch die *WebMvcConfigurerAdapter* Klasse vererbt.

Die Konfigurationsklasse für den Datenbankzugriff beinhaltet zum Einen Einstellungen zur Datenbank, wie Benutzer und JDBC-Treiber und zum Anderen aber auch die Entities verwaltende Instanz, die sogenannte *LocalContainerEntityManagerFactoryBean*. Mit dem *JpaTransactionManager*, welcher für

das Verwalten der Transaktionen mit der Datenbank zuständig ist und dem Setzen von Hibernate als verwendeten Persistence-Layer, sind die Kernaufgaben dieser Konfigurationsklasse abgeschlossen. Zudem wird noch ein einfacher Boolean über diese Klasse im Applikationskontext registriert. Über diesen wird erfasst, ob es sich um eine gateway- oder datacenterspezifische Datenbank handelt. Jegliche in dieser Klasse für die Konfiguration notwendigen Attribute, wie etwa der zu verwendende JDBC-Treiber oder die Adresse der Datenbank, werden in einer Java-Konfigurationsdatei, einem sogenannten *Properties File*, hinterlegt. Über ein sogenanntes *Environment* Objekt, über welches man auf umgebungsrelevante Daten Zugriff erlangt, kann dieses in der Applikation ausgelesen werden. So ist es möglich über das Anpassen der Attribute im Properties File die verwendete Datenbank und die damit einhergehend benötigten Konfigurationsattribute einfach zu ändern. Um diese in den laufenden Applikationskontext zu laden, ist lediglich ein erneutes Laden des Applikationskontextes notwendig. Dafür wird eine eigene die eingegebenen Datenbankzugriffseinstellungen validierende und eine das Properties File aktualisierende Klasse implementiert.

Die Authentifizierung für den Zugriff auf die SmartVis Webapplikation wird mit Spring Security implementiert. Die Nutzer- und Rolleninformationen befinden sich in der MySQL-Datenbank.

### 2.3 Build- und Dependency-Management Maven

Für das Build- und Dependency-Management wird die Software *Maven* der Apache Software Foundation eingesetzt. Es abstrahiert den Buildprozess und vereinfacht durch Automatisierung häufig anfallende Ausführungsschritte in der Softwareentwicklung. Somit ist eine Unabhängigkeit von jeglicher Entwicklungsumgebung gegeben.

Dem Prinzip "Konvention über Konfiguration" folgend, sieht Maven eine Standardstrukturierung eines Softwareprojekts vor. Dies hat mehrere Vorteile: Zum einen ist ein Wiedererkennungseffekt gegeben, wodurch sich auch nicht mit dem Softwareprojekt Vertraute schnell in der Paketstruktur des Projekts zurecht finden. Zudem sind kaum Konfigurationen vorzunehmen, da Maven bereits weiß, wo es innerhalb des Projekts nach welchen Komponenten suchen muss, vorausgesetzt man hält sich an die Struktur die Maven vorgibt. Maven bietet einem auch die Möglichkeit so viel wie gewünscht selbst zu konfigurieren, jedoch geht dann ein wesentlicher Vorteil dieses Prinzips verloren.

Darüber hinaus ist einer der größten Vorteile von Maven das Übernehmen der Verwaltung von Abhängigkeiten zwischen benötigten Bibliotheken und Paketen innerhalb eines Projekts. Maven versucht dabei auf Seiteneffekte von unterschiedlichen Bibliotheken Rücksicht zu nehmen und so Abhängigkeiten gleich mit zu laden. Dies wird durch Archive, sogenannte *Repositories*, bewerkstelligt, in denen Maven nach den benötigten Bibliotheken und Paketen sucht. Hierbei unterscheidet man zwischen *Local*, *Remote* und *Central Repositories*. In das *Local Repository* lädt Maven alle Abhängigkeiten eines Projekts um sie lokal zur Verfügung zu haben. *Remote Repositories* stellen benutzerdefinierte Archive dar, in denen man Bibliotheken und Pakete für Maven hinterlegen kann, zum Beispiel um firmenweit eine bestimmte Auswahl an Bibliotheken zur Verfügung zu stellen. Das *Central Repository* ist ein von der Maven Community zur Verfügung gestelltes offizielles Archiv in dem man die meisten Bibliotheken findet.

Das Einbinden der benötigten Abhängigkeiten als auch das Setzen von Konfigurationen für die unterschiedlichen Phasen des Entwicklungsprozesses, werden im sogenannten *Project-Object-Model*, kurz *POM-File* erledigt. Es stellt somit die Konfigurationsdatei eines Maven-Projekts dar.

### 3 Konzeption der Webschnittstelle

Um die Datenabfrage vom Client aus auf eine einheitliche und strukturierte Art und Weise zu ermöglichen, wurden Schnittstellenbeschreibungen für die Webservices konzipiert. Dabei wurde darauf Wert gelegt die Meter- und Metadatenchnittstellen RESTful anzulegen.

Das Webservice übergibt in der Aufruf-URL die Request-Parameter und gibt immer JSON-Objekte an den Aufrufer (Browser) zurück. Timestamp-Angaben erfolgen im Format ISO 8601 (zum Beispiel: 2013-04-01T13:01:02).

#### Interface Messdatenabfrage

Dieses Web-Service dient zur Abfrage von Smart Meter Messdaten und hat folgenden URL-Aufbau:

<http://{Hostname}:{Portnummer}/smartvis/{dbConnection}/meterdata/{meterId}/{channel}/{from}/{to}/{resolution}/{intervall}>

Beschreibung der Parameter:

Attribut	Format	Beschreibung																																																												
dbConnection	Integer	ID für DB-Verbindung (1 ... n)																																																												
meterId	Integer	ID des Smart Meter; 0...Wert um statt eines Meters das HO-Profil abzurufen																																																												
channel	String	<table> <tr> <th>Wert</th><th>Beschreibung</th><th>Aggregation</th></tr> <tr><td>Counter</td><td>Zählerstand gesamt [KWh]</td><td>MAX</td></tr> <tr><td>counter1</td><td>Stand Zähler1 [KWh]</td><td>MAX</td></tr> <tr><td>counter2</td><td>Stand Zähler2 [KWh]</td><td>MAX</td></tr> <tr><td>counter3</td><td>Stand Zähler4 [KWh]</td><td>MAX</td></tr> <tr><td>counter4</td><td>Stand Zähler4 [KWh]</td><td>MAX</td></tr> <tr><td>consumption</td><td>Stromverbrauch gesamt [Wh]</td><td>SUM</td></tr> <tr><td>consumptionP1</td><td>Stromverbrauch P1 [Wh]</td><td>SUM</td></tr> <tr><td>consumptionP2</td><td>Stromverbrauch P2 [Wh]</td><td>SUM</td></tr> <tr><td>consumptionP3</td><td>Stromverbrauch P3 [Wh]</td><td>SUM</td></tr> <tr><td>power</td><td>Leistung gesamt [W]</td><td>AVG</td></tr> <tr><td>powerP1</td><td>Leistung P1 [W]</td><td>AVG</td></tr> <tr><td>powerP2</td><td>Leistung P2 [W]</td><td>AVG</td></tr> <tr><td>powerP3</td><td>Leistung P3 [W]</td><td>AVG</td></tr> <tr><td>current</td><td>Strom gesamt [A]</td><td>AVG</td></tr> <tr><td>currentP1</td><td>Strom P1 [A]</td><td>AVG</td></tr> <tr><td>currentP2</td><td>Strom P2 [A]</td><td>AVG</td></tr> <tr><td>currentP3</td><td>Strom P3 [A]</td><td>AVG</td></tr> <tr><td>Voltage</td><td>Spannung [V]</td><td>AVG</td></tr> <tr><td>Frequency</td><td>Frequenz [Hz]</td><td>AVG</td></tr> </table>	Wert	Beschreibung	Aggregation	Counter	Zählerstand gesamt [KWh]	MAX	counter1	Stand Zähler1 [KWh]	MAX	counter2	Stand Zähler2 [KWh]	MAX	counter3	Stand Zähler4 [KWh]	MAX	counter4	Stand Zähler4 [KWh]	MAX	consumption	Stromverbrauch gesamt [Wh]	SUM	consumptionP1	Stromverbrauch P1 [Wh]	SUM	consumptionP2	Stromverbrauch P2 [Wh]	SUM	consumptionP3	Stromverbrauch P3 [Wh]	SUM	power	Leistung gesamt [W]	AVG	powerP1	Leistung P1 [W]	AVG	powerP2	Leistung P2 [W]	AVG	powerP3	Leistung P3 [W]	AVG	current	Strom gesamt [A]	AVG	currentP1	Strom P1 [A]	AVG	currentP2	Strom P2 [A]	AVG	currentP3	Strom P3 [A]	AVG	Voltage	Spannung [V]	AVG	Frequency	Frequenz [Hz]	AVG
Wert	Beschreibung	Aggregation																																																												
Counter	Zählerstand gesamt [KWh]	MAX																																																												
counter1	Stand Zähler1 [KWh]	MAX																																																												
counter2	Stand Zähler2 [KWh]	MAX																																																												
counter3	Stand Zähler4 [KWh]	MAX																																																												
counter4	Stand Zähler4 [KWh]	MAX																																																												
consumption	Stromverbrauch gesamt [Wh]	SUM																																																												
consumptionP1	Stromverbrauch P1 [Wh]	SUM																																																												
consumptionP2	Stromverbrauch P2 [Wh]	SUM																																																												
consumptionP3	Stromverbrauch P3 [Wh]	SUM																																																												
power	Leistung gesamt [W]	AVG																																																												
powerP1	Leistung P1 [W]	AVG																																																												
powerP2	Leistung P2 [W]	AVG																																																												
powerP3	Leistung P3 [W]	AVG																																																												
current	Strom gesamt [A]	AVG																																																												
currentP1	Strom P1 [A]	AVG																																																												
currentP2	Strom P2 [A]	AVG																																																												
currentP3	Strom P3 [A]	AVG																																																												
Voltage	Spannung [V]	AVG																																																												
Frequency	Frequenz [Hz]	AVG																																																												
from	Timestamp ISO 8601	Anfangszeitpunkt																																																												
to	Timestamp ISO 8601	Endzeitpunkt																																																												

resolution	String	<div>zeitliche Auflösung</div> <div>Werte:</div> <table><thead><tr><th>Auflösung</th><th>Beschreibung</th></tr></thead><tbody><tr><td>All</td><td>alle Daten - ohne Aggregation</td></tr><tr><td>minute</td><td>Aggregation auf Minuten</td></tr><tr><td>Hour</td><td>Aggregation auf Stunden</td></tr><tr><td>Day</td><td>Aggregation auf Tage</td></tr><tr><td>month</td><td>Aggregation auf Monate</td></tr><tr><td>year</td><td>Aggregation auf Jahre</td></tr><tr><td>Last</td><td>Zeitlich letzter Wert</td></tr></tbody></table>	Auflösung	Beschreibung	All	alle Daten - ohne Aggregation	minute	Aggregation auf Minuten	Hour	Aggregation auf Stunden	Day	Aggregation auf Tage	month	Aggregation auf Monate	year	Aggregation auf Jahre	Last	Zeitlich letzter Wert
Auflösung	Beschreibung																	
All	alle Daten - ohne Aggregation																	
minute	Aggregation auf Minuten																	
Hour	Aggregation auf Stunden																	
Day	Aggregation auf Tage																	
month	Aggregation auf Monate																	
year	Aggregation auf Jahre																	
Last	Zeitlich letzter Wert																	
interval	Integer	<div>Bei Aggregation: Anzahl der zu aggregierenden Einheiten</div> <div>zum Beispiel bei resolution=minute, interval=5 -&gt; 5 Minuten werden zusammengefasst</div> <div>Das Intervall ist Auflösungsabhängig:</div> <table><thead><tr><th>Auflösung</th><th>mögliche Intervall-Werte</th></tr></thead><tbody><tr><td>all</td><td>0</td></tr><tr><td>minute</td><td>1, 5, 15, 30</td></tr><tr><td>hour</td><td>1, 4, 6, 12</td></tr><tr><td>day</td><td>1, 7, 14, 21</td></tr><tr><td>month</td><td>1, 3, 6</td></tr><tr><td>year</td><td>1</td></tr></tbody></table>	Auflösung	mögliche Intervall-Werte	all	0	minute	1, 5, 15, 30	hour	1, 4, 6, 12	day	1, 7, 14, 21	month	1, 3, 6	year	1		
Auflösung	mögliche Intervall-Werte																	
all	0																	
minute	1, 5, 15, 30																	
hour	1, 4, 6, 12																	
day	1, 7, 14, 21																	
month	1, 3, 6																	
year	1																	

Tabelle 1: Web-Service - Schnittstellenparameter Messdatenabfrage

Die konzipierte URL für die Abfrage deckt mit ihren Parametern sowohl das Ansprechen unterschiedlicher Datenbanken, wie es für die Präsentationssicht notwendig ist, ab, als auch unterschiedliche Meter, repräsentiert durch ihre Meter-Id.

Darüber hinaus kann zwischen unterschiedlichen Messwerten des Meters, wie zum Beispiel dem Zählerstand, der Leistung, der Spannung, dem Strom oder der Frequenz mittels sogenannter Kanäle zugegriffen werden. Aber auch von der Applikation aggregierte Werte wie die Leistung können so erfragt werden.

Nach Angabe der Art der gewünschten Informationen durch den Kanal, wird der Zeitraum angegeben, für den man die Werte möchte. Dieser wird durch Zeitinformationen im Timestamp-Format laut ISO-8601 abgefragt. Schließlich wird die gewünschte Auflösung angegeben, gefolgt vom Intervall in dem die Daten schlussendlich aggregiert werden sollen. Mit Auflösungen von jedem einzelnen Wert über eine Minute, Stunde, Tag, Monat, hin zu einem Jahr, steht, im Zusammenhang mit weiteren unterschiedlichsten Intervallmöglichkeiten, eine Vielzahl an Granularitätsstufen zur Verfügung. Die Art der Intervalle ist auflösungsabhängig.

Das Aufrufbeispiel (siehe Abbildung 1) zeigt die Anfrage an die Datenbank 0 um Meterdaten des Meters mit der Id 2. Dabei werden Spannungswerte im Zeitraum vom 24.11.2014 um 15 Uhr bis zum 24.11.2014 um 15:01 Uhr abgefragt. Diese sollen in voller Auflösung mit einem Intervall von 0 ausgegeben werden, was bedeutet, dass jeder einzelne Wert in diesem Zeitraum dargestellt werden soll.

```

/0/meterdata/2/voltageP3/2014-11-24T15:00:00.000Z/2014-11-24T15:00:59.000Z/all/0

[[{"2014-11-24T15:00:01",235.7},
{"2014-11-24T15:00:01",235.7},
{"2014-11-24T15:00:10",235.6},
{"2014-11-24T15:00:20",235.9},
{"2014-11-24T15:00:30",236.1},
{"2014-11-24T15:00:40",235.9},
{"2014-11-24T15:00:50",235.5}]]

```

Abbildung 1: Meterdaten-Schnittstellenaufwurf und Response-Daten

### Interface Meter-Metadaten

Dieses Web-Service liefert Meta-Daten zu den vorhandenen Smart Meter Geräten zurück (Rückgabe GW und DC sind ident).

Aufruf-URL	<a href="#">/{dbConnection}/metermetadata/</a>
Aufrufbeispiel	<a href="http://localhost:8080/0/metermetadata">http://localhost:8080/0/metermetadata</a>
Response-Daten	<pre> [   {     id: 1,     description: "MT830-3080",     serial: "35770817",     period: 15,     type: "Electric",     manufactor: "Iskraemeco",     customerid: 1,     firstname: "Max",     lastname: "Mustermann",     counter: ""   } ] </pre>

Tabelle 2: REST-Schnittstelle metermetadaten

### Interface SmartMeter-Events

Dieses Web-Service liefert Event-Daten zu den vorhandenen Smart Meter Geräten zurück (Rückgabe GW und DC sind ident).

Aufruf-URL	<a href="#">/{dbConnection}/meterevents/{meterId}/{eventType}/{from}/{to}</a>
Aufrufbeispiel	<a href="http://localhost:8080/0/meterevent/1/1/2015-12-14T00:00:00.000Z/2016-01-01T23:00:00.000Z">http://localhost:8080/0/meterevent/1/1/2015-12-14T00:00:00.000Z/2016-01-01T23:00:00.000Z</a>
Response-Daten	<pre> [   {     logId: 1,     idType: 1,     content: "Test_Content",     sourceTarget: "Test_Source",     timestamp: 1426032000000   } ] </pre>

Tabelle 3: REST-Schnittstelle meterevents

Gültige Werte für "eventType":

- 1 = error
- 2 = log



## 4 Anwendungskomponenten

Nachfolgend werden die wesentlichen Komponenten der SmartVis-Anwendung beschrieben.

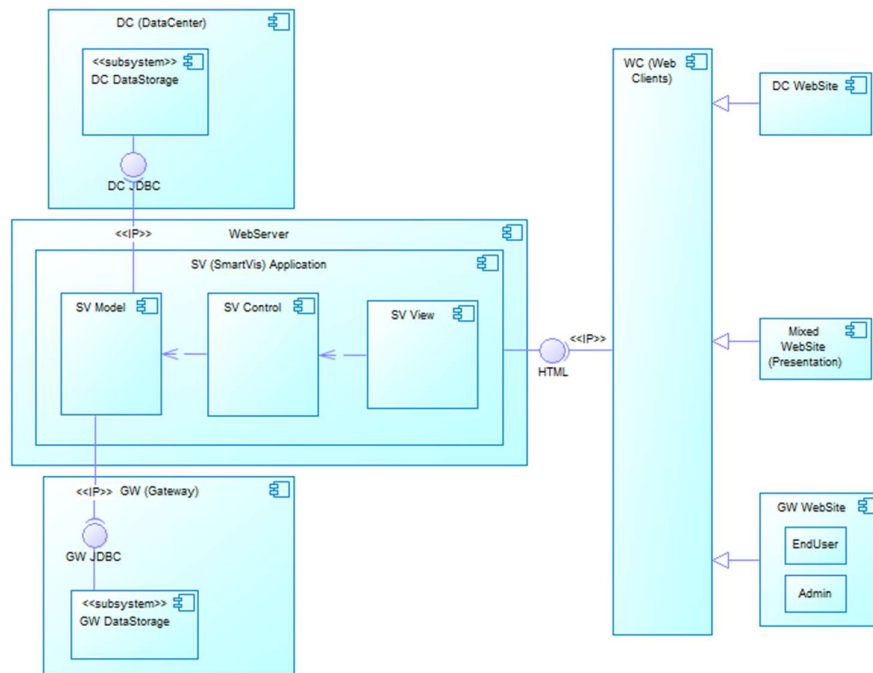


Abbildung 2: Komponenten SmartVis

### 4.1 Model

In diesem Unterabschnitt wird das „Model“ (vgl. MVC-Pattern) für die SmartVis Implementierung beschrieben.



Abbildung 3: Schichten im SmartVis Modell

In Abbildung 3 ist die Struktur des SmartVis Models dargestellt: Im *Service Layer* befinden sich aufgabenbezogene Service-Klassen, die auf Repository-Klassen im *Persistence Layer* zugreifen um Objekte zu speichern oder aus dem Speicher abzufragen. Repository-Klassen wiederum verwenden in der vorliegenden Implementierung Entity-Klassen, um eine zusätzliche Abstraktion der Daten zu erzielen. Eine Trennung von Service Layer und Persistence Layer wurde gewählt, um Anwendungslogik von Datenzugriff beziehungsweise Datenbereitstellung sauber zu trennen.

Die Bereitstellung der Smart Meter Messdaten ist eine Kernkomponente der SmartVis Anwendung. Aus diesem Grund wird nachfolgend die diesbezügliche Implementierung im Detail ausgeführt.

#### 4.1.1 Service Layer

Im Service Layer befinden sich aufgabenbezogene Service-Klassen, die von MVC-Controllern instanziiert werden. Die Bereitstellung und Aufbereitung von Messdaten eines Smart Meter Geräts erfolgt in der Klasse *MeterDataServiceImpl*.

Die Methode *aggregatedMeterData* fragt Messwerte vom Persistence Layer ab (siehe Listing 1):

```
public List<Object[]> aggregatedMeterData(int dbSource, int meterId,
    String channel, Timestamp startStamp, Timestamp endStamp, String
    resolution, int interval) { ... }
```

Listing 1: Signatur Methode *aggregatedMeterData*

Die verwendeten Übergabeparameter zur Selektion und Aggregation der Abfragedaten werden vom Web-Client vorgegeben und sind so gewählt, dass nur eine einzige Schnittstellenvereinbarung für alle möglichen Varianten der Messdaten-Abfrage erforderlich ist. Der Rückgabewert ist unabhängig vom konkreten Messwert als Objekt-Liste (*List<Object[]>*) definiert (siehe Listing 1). Das Ergebnis für die Datenübermittlung an den Web-Client soll einen möglichst schlanken Aufbau aufweisen, weil teils beträchtliche Datenmengen verarbeitet werden. Die Ergebnisdaten werden daher von der rufenden Controller-Klasse in weiterer Folge in JavaScript Object Notation (JSON) konvertiert und an den Web-Client gesendet.

Wenn die Ausgabe von Stromverbrauchsdaten vom Client angefordert wurde, dann wird die Methode *calculateConsumption* zur Ermittlung dieser Daten aufgerufen (siehe Listing 2)

```
public static List<Object[]> calculateConsumption
    (List<Object[]> inList, Timestamp startStamp)
```

Listing 2: Signatur Methode *calculateConsumption*

Die Stromverbrauchsdaten sind in der benötigten Form in der Datenbank nicht enthalten. Der Zählerstand je Messzeitpunkt wird zwar in verschiedenen „Count“-Datenfeldern gespeichert, allerdings liegt diese Information ausschließlich als ganzzahliger Wert in Kilowattstunden vor. Somit lassen sich die Stromverbrauchswerte nicht in der erforderlichen Detaillierung extrahieren.

Die Methode *calculateConsumption* iteriert durch die Objekte aus der Rückgabeliste *aggregatedMeterData*. Es wird die Dauer zwischen zwei Messzeitpunkten ermittelt und mit der Leistung in Watt [W] multipliziert:

```
tmpObj[1] = (meterValue * diff_in_msec) / 3600000;
```

Das Ergebnis entspricht dem Verbrauchswert in Wattstunden [Wh] und wird zusammen mit dem Messzeitpunkt wieder in eine Objekt-Liste eingetragen.

Die Visualisierung der Smart Meter Messwerte muss in zeitlichen Granulierungsstufen erfolgen, die durch Aggregierungsfunktionen der Datenbank-Abfrage nicht in allen Fällen abgebildet werden kann.

Wenn beispielsweise der Stromverbrauch in Intervallen zu je 5 Tagen summiert werden muss, dann kann die Datenbank mit der SUM-Aggregierungsfunktion Summen je Tag bilden. Die Aufsummierung von 5 Tagen muss jedoch softwareseitig erfolgen.

In der vorliegenden Implementierung wird diese softwareseitige Aggregierung durch die Methode *aggregateToIntervall* realisiert:

```
public static List<Object[]> aggregateToInterval  
    (List<Object[]> inList, String channel,  
     String resolution, int interval)
```

Listing 3: Signatur Methode *aggregateToInterval*

Der Methode *aggregateToInterval* (siehe Listing 3) werden als Übergabeparameter die Liste der Messwerte sowie die Informationen *channel*, *resolution* und *interval* übergeben. Auf Basis dieser Parameter werden in der Methode dynamisch Zeitintervalle gebildet und Messwerte gruppiert und aggregiert. Das Ergebnis wird in Form einer Objekt-Liste retourniert.

#### 4.1.2 Persistence Layer

Ein wesentliches Ziel der SW-Lösung ist die Kapselung des Datenzugriffs, so dass Änderungen in der Datenhaltung ohne großen Entwicklungsaufwand realisiert werden können. Im Java Standard wird dieses Ziel durch die Java Persistence API (JPA) unterstützt. Im Kern sieht diese Schnittstelle die Zuordnung und Speicherung (Persistierung) von Objektinstanzen in relationalen Datenbanken vor. Zwei Komponenten sind dazu wesentlich:

- Persistence Entities: dabei handelt es sich um Java Objektinstanzen, die eine vorgegebene Struktur aufweisen müssen und letztendlich als Datensätze in Datenbanktabellen gespeichert werden.
- Objekt-Relationales Mapping (ORM): damit wird die Zuordnung von Persistence Entities zu Datenbankrelationen definiert.

Die tatsächliche Interaktion mit der jeweiligen Datenbank erfolgt dabei durch eine frei wählbare Middleware. JPA-Aufrufe werden von diesen „Persistence Providern“ in (für die jeweilige Datenbank) korrekte SQL-Anweisungen „übersetzt“. In der vorliegenden Lösung wird dazu das Open-Source-Framework Hibernate verwendet. Hibernate wiederum nutzt eine JDBC-Schnittstelle um auf die bestehenden MySQL-Daten zuzugreifen. Der implementierte Technologie-Stack wird in Abbildung 4 dargestellt:

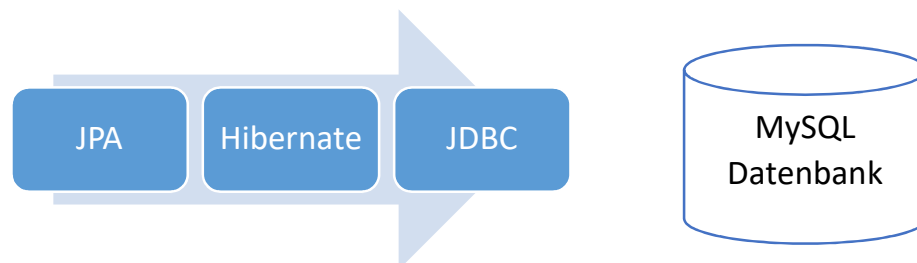


Abbildung 4 Ablauf Datenzugriff SmartVis

Moderne Entwicklungstools unterstützen die Erstellung beziehungsweise Codierung von Entity-Klassen. Auf Basis von Datenbankbeschreibungen können diese Klassen automatisch generiert werden. Entity Klassen werden mit `@Entity` annotiert und müssen die eindeutige Identifizierung der Objektinstanz mit der Annotation `@Id` vorsehen.

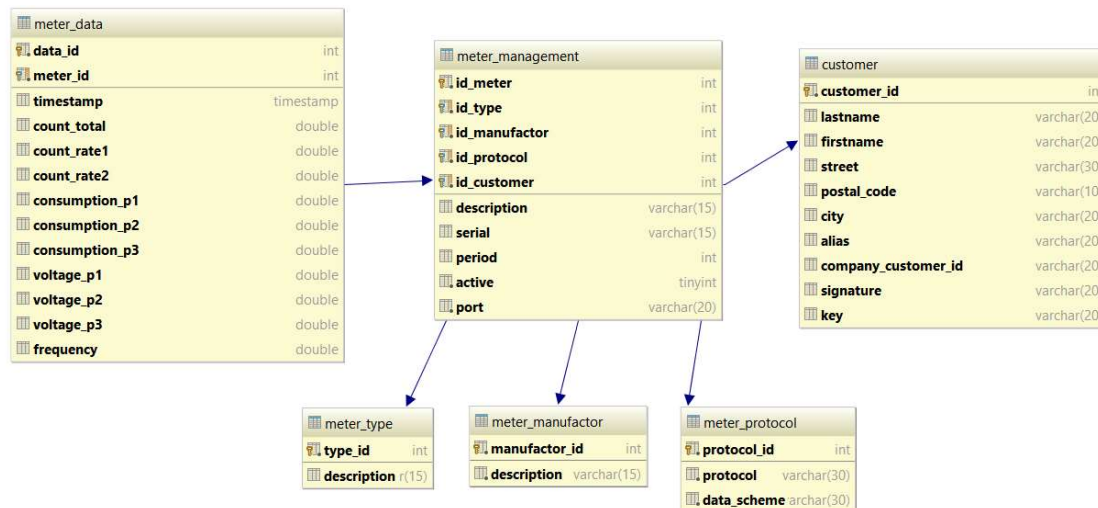


Abbildung 5: ER-Diagramm Meter-Daten

Der DB-Aufbau von Datacenter und Gateway ist bezüglich der zu visualisierenden Daten ident. Für die Datenbereitstellung wurden Funktionen mit dem Ziel erstellt, Aufrufe und Rückgabewerte unabhängig von der Art des Quellsystems (GW oder DC) zu gestalten. Dies betrifft zum einen Smart Meter Messdaten und zum anderen Meta-Informationen zu den Smart Meter Geräten. Des Weiteren waren die Datenzugriffe im Fall der „Show-Sicht“ nicht auf eine Datenbankverbindung beschränkt.

#### Abfrage von Smart Meter Messdaten

Die Abfrage der Messdaten wird in der Klasse *MeterDataRepository* mit der Methode *selectMeterData* realisiert. In Listing 4 ist die Methode dargestellt.

```

public List<Object[]> selectMeterData(int dbSource,           1)
                                   int meterId,
                                   String channel,
                                   Date startDate,
                                   Time startTime,
                                   Date endDate,
                                   Time endTime,
                                   String resolution) {

if (dbSource==0 )           2)
    entityManager = db1EntityManagerFactory.createEntityManager();
else
    entityManager = db2EntityManagerFactory.createEntityManager();

Query query = entityManager.createQuery(getMeterDataSelectQuery());  3)
query.setParameter("meterId", meterId);
query.setParameter("startDate", startDate);
query.setParameter("startTime", startTime);
query.setParameter("endDate", endDate);
query.setParameter("endTime", endTime);

List<Object[]> result=query.getResultList();           4)
entityManager.close();
return result;
}
  
```

Listing 4: Methode selectMeterData für die Abfrage von Meter-Daten

Nachfolgend werden die Programmteile aus Listing 4 beschrieben:

ad 1) Die Methodensignatur enthält als Eingabewerte die Selektionskriterien für die Abfrage.

ad 2) Der *entityManager* verwaltet den Zugriff auf Persistence Entities. Anhand der Variablen *dbSource* wird der benötigte *entityManager*, und damit implizit die benötigte Datenbankverbindung gewählt.

ad 3) Der Zugriff auf die Entities erfolgt mit einem Query. Der Query-String wird in der Methode *getMeterDataSelectQuery* dynamisch auf Basis der Parameterwerte generiert.

ad 4) Der Rückgabewert *List<Object[]>* ist auf die minimal benötigten Daten reduziert: es wird eine Liste mit je zwei Daten-Objekten ausgegeben: der Zeitpunkt der Messung sowie der jeweilige Messwert (siehe Listing 5).

```
[["2014-06-11T00:00:01",0.1],
["2014-06-11T00:00:49",0.1],
["2014-06-11T00:01:36",0.1],
["2014-06-11T00:02:24",0.1],
["2014-06-11T00:03:11",0.1],
["2014-06-11T00:03:58",0.1],
["2014-06-11T00:04:46",0.1]]
```

Listing 5: Exemplarische JSON-Rückgabewerte der Methode *selectMeterData*

In JPA erfolgt die Abfrage von Persistence Entities mit Java Persistence Query Language (JPQL). Die JPQL-Notation ist ähnlich wie SQL, allerdings fließen Konzepte der Objektorientierung in die Abfrage ein (siehe Listing 6).

```
select (concat(m.date, 'T', m.time))
from GW_MeterDataEntity m
where m.idMeter = :meterId
and (m.date || m.time) between (:startDate || :startTime)
and (:endDate || :endTime)
```

Listing 6: JPQL-Abfrage zu Ergebnisanzeige in Listing 5

Der JPA-Provider Hibernate generiert aus der JPQL-Abfrage die SQL-Anweisung für den JDBC-Zugriff auf die MySQL-Datenbank:

```
select concat(gw_meterda0_.date, 'T', gw_meterda0_.time) as col_0_0_,
       gw_meterda0_.consumption_p1+gw_meterda0_.consumption_p2+gw_meterda0_.consumption_p3 as col_1_0_
from meter_data gw_meterda0_
where gw_meterda0_.id_meter=? and (concat(gw_meterda0_.date,
       gw_meterda0_.time) between concat(?, ?) and concat(?, ?))
```

Listing 7: von Hibernate generierte SQL-Abfrage zu JPQL-Abfrage in Listing 6

## Spring Repositories

Die im vorigen Unterabschnitt beschriebene Implementierung der Datenzugriffe war bei der SmartVis Implementierung nicht die präferierte Wahl. Vielmehr war die Nutzung von *JpaRepositories* aus dem Framework „Spring Data JPA“ [1] geplant, da diese Technologie den Codierungsaufwand für Persistierungsfunktionen reduzieren kann und soll. Auf eine ausführlichere Beschreibung der Konzepte und Funktionen von Spring Data JPA wird an dieser Stelle verzichtet.

```
public interface DCMeterRepository
extends JpaRepository<DC_MeterDataEntity, Integer>
{
    @Query("select concat(m.date, 'T', m.time), (m.consumptionSum*233) " +
           " from DC_MeterDataEntity m " +
```

```
" where m.idMeter = :meterId and " +
"and (m.date || m.time) between (:startDate || :startTime)" +
"and (:endDate || :endTime)";

List<Object[]> powerAll (@Param("meterId") int meterId,
                        @Param("startDate") Date startDate,
                        @Param("startTime") Time startTime,
                        @Param("endDate") Date endDate,
                        @Param("endTime") Time endTime);
}
```

Listing 8: Spring Data JPA - Beispiel JpaRepository

In Listing 8 wird beispielhaft ein *JpaRepository* angeführt: die Codierung des Interface *DCMeterRepository* ist ausreichend, um eine JPQL-Datenabfrage auszuführen. Aus zwei Gründen wurde diese Technologie für die Abfrage von Messdaten nicht angewendet:

- Die @Query-Annotation schränkt die dynamische Erzeugung von Query-Strings stark ein. Das hat zur Folge, dass für die unterschiedlichen Messdaten-Abfragen eine Vielzahl von Methoden mit sich wiederholenden Code-Strecken benötigt wurde.
- Die SmartVis Anwendung benötigt im Fall der „Show-View“ Messdaten aus zwei verschiedenen Datenbanken. Die dynamische Zuordnung der jeweils benötigten Datenbank-Verbindung wurde mit den dafür vorgesehenen „managed Transactions“ nicht hergestellt. Stattdessen wurde die erste aufgebaute Verbindung immer beibehalten und für alle folgenden Datenbankzugriffe einer Persistence Entity verwendet. Ob dieses Verhalten durch eine fehlerhafte Implementierung oder Konfiguration von SmartVis hervorgerufen wurde, konnte nicht festgestellt werden.

## 4.2 View

In diesem Unterabschnitt wird auf die Visualisierung, die Umsetzung des Frontends und die verwendeten HTML/CSS und JavaScript-Frameworks eingegangen.

### 4.2.1 HTML Templates

Als Framework für die HTML-Templates wird für Bootstrap von Twitter verwendet. Dieses bietet bereits vordefinierte HTML-Elemente wie Buttons, Navigationselemente, Dialoge und vieles mehr, welches sich einfach und ohne viel Aufwand integrieren lässt. Mithilfe des Gridsystems lassen sich responsive Webseiten erstellen, die sich an die Browserbreite dynamisch anpassen und somit eine optimale Bedienbarkeit für alle Endgeräte gewährleistet wird. Die klassischen GUI-Elemente sind dabei mithilfe der Bootstrap-CSS schon formatiert und können direkt verwendet werden, was eine erhebliche Arbeitserleichterung und Zeitersparnis bedeutet.

Das Layout wurde sehr einfach und übersichtlich gehalten. Die Webseite lässt sich in drei Bereiche einteilen:

- Header
- Hauptmenü
- Contentbereich

Der Header entspricht dabei lediglich einer Leiste, die den Webseiten-Titel beinhaltet. Für kleine Displays wird rechts ein Button eingeblendet, mit dem man das Hauptmenü ein- und ausklappen kann.

Das Hauptmenü befindet sich links neben dem Contentbereich und ist bei größeren Displays immer sichtbar. Bei kleineren Displays wird das Hauptmenü versteckt, um so den Contentbereich zu maximieren. Das Hauptmenü wird nach links eingeklappt und ist nicht scrollbar. Dies bietet gerade bei mobilen Endgeräten eine bessere Usability, da man bei horizontal ausklappbaren Menüs oft nach oben scrollen muss, um das Menü zu sehen, wenn man sich bereits weiter unten befindet.

Eine vereinfachte Darstellung der Webseite und das Mockup vom Seitenlayout ist in Abbildung 6 zu sehen.

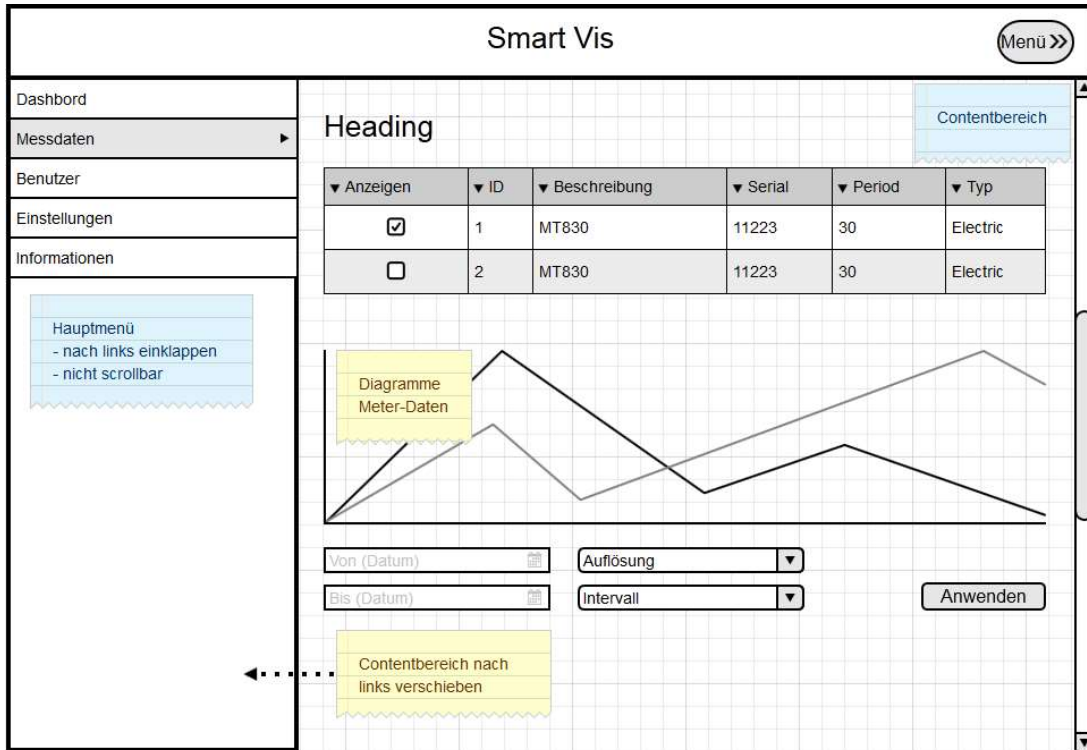


Abbildung 6: Mockup des Seitenlayouts

Um das Bootstrap-Framework verwenden zu können, muss lediglich die Bootstrap-CSS und die Bootstrap-JavaScript eingebunden werden. Das Bootstrap-Framework ist über eine Webjar verfügbar, was die Integration über Maven ermöglicht. Danach lassen sich die benötigten Dateien vom Webjars-Ordner einbinden. Da die Webseite auf einem Tomcat-Server läuft, werden für die HTML-Templates JavaServer Pages verwendet. Um die JavaServer Pages Standard Tag Library (JSTL) zu verwenden, muss diese zu Beginn des Dokumentes angegeben werden.

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>

<!DOCTYPE html>
```

Abbildung 7: Einbindung JSTL-Tag Library

```
<!-- Bootstrap CSS -->
<link href="<c:url value='/'>webjars/bootstrap/3.3.0/css/bootstrap.min.css" rel="stylesheet" media="screen" />
```

Abbildung 8: Einbindung Bootstrap-CSS über Webjars-Ordner



Um das gewünschte Layout umzusetzen, wird das Gridsystem von Bootstrap verwendet. Damit lassen sich die Inhalte und Elemente auf der Webseite in einem Raster anordnen, welches sich je nach Bildschirmbreite anpasst. Ein weiterer Vorteil ist, dass alle Elemente und Inhalte sich in einer Linie befinden oder anders ausgedrückt, zueinander ausgerichtet sind. Das Gridsystem besteht aus maximal 12 Spalten, welche sich beliebig kombinieren lassen.

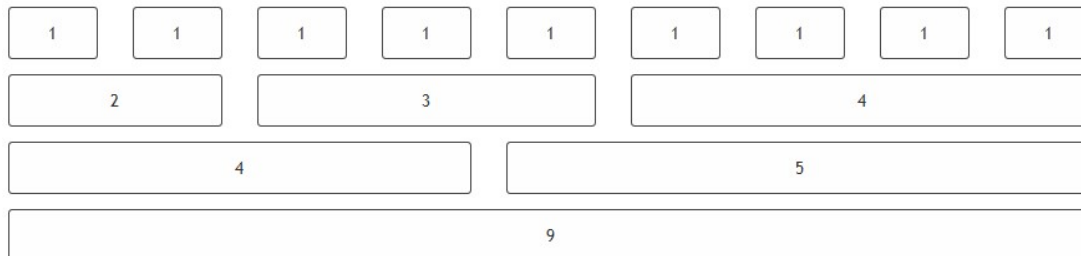


Abbildung 9: Bootstrap Gridsystem [2]

Jede Zeile wird mit der CSS-Klasse *row* definiert. Für die Spalten kann man die Column-Klassen verwenden, die wie folgt aufgebaut sind: *.col-size-n*

Für *size* kann man dabei die Werte *xs*, *sm*, *md*, *lg* einsetzen. Für *n* eine Zahl zwischen 1-12, diese gibt die Spaltenbreite an wie in Abbildung 10 abgebildet. *Size* definiert dabei die Bildschirmgrößen, ab welcher diese Eigenschaft aktiv wird, wobei diese aufwärts kompatibel ist, was bedeutet dass zum Beispiel *xs* auch auf größeren Bildschirmen aktiv ist, solange diese nicht überschrieben wird. Somit lassen sich für verschiedene Bildschirmgrößen zum Beispiel unterschiedliche Spaltenanzahlen definieren:

```
<div class="row">
  <div class="col-xs-1 col-sm-2 col-md-4"></div>
</div>
```

Abbildung 10: Beispiel Definition versch. Spalten für versch. Bildschirmgrößen

So würde der Div-Container bei sehr kleinen Bildschirmen wie zum Beispiel bei einem Smartphone eine Spaltenbreite von 1 besitzen. Bei größeren Bildschirmen wie zum Beispiel bei einem Tablet eine Spaltenbreite von 4. Nach diesem Prinzip lassen sich alle denkbaren Layouts responsiv, also Layouts die sich an die Browser- und Bildschirmbreite anpassen, umsetzen.

Bestimmte Bereiche, die immer wieder verwendet werden, wurden in eigene JSP-Seiten ausgelagert. Der Headerbereich als Beispiel ändert sich pro Seite nicht und kann daher auf jeder Seite wieder verwendet werden. Dasselbe gilt für das Hauptmenü. Mithilfe der *jsp:include* Funktion lassen sich diese Bereiche dann an einer beliebigen Stelle im Template einfügen. Somit minimiert man den Code und gewährleistet eine gute Wartbarkeit, da Änderungen nur an einer Stelle durchgeführt werden müssen.

#### 4.2.2 Login

Um auf die Webseite zugreifen zu können, muss man sich zuerst authentifizieren. Dafür wird ein Login-Formular angezeigt, sobald man die Webseite im Browser öffnet. Die Authentifizierung wurde serverseitig mithilfe von Spring-Security umgesetzt. Dafür wird eine Klasse *SecurityConfig* definiert, die von der Klasse *WebSecurityConfigurerAdapter* erbt. Mithilfe dieser Konfigurationsklasse kann man Benutzer, Passwörter und Benutzergruppen definieren. Benutzernamen und Passwörter sind in der MySQL-Datenbank gespeichert.



### 4.2.3 Benutzergruppen und Rechte

Es gibt verschiedene Benutzergruppen, die unterschiedliche Rechte besitzen. Bestimmte Funktionen und Seiten sollen nur für bestimmte Benutzergruppen sichtbar und aufrufbar sein. Im Template lässt sich die aktuelle Benutzergruppe mithilfe der Spring Security *authorize* Funktion abfragen, wie in Abbildung 11 gezeigt:

```
<sec:authorize access="hasRole('ROLE_ADMIN')">
  <c:url value="/volt-power" var="urlVoltPower" />
  <li class="<c:if test="{urlVoltPower == currentSite}">active</c:if">
    <a href="{urlVoltPower}">Spannung & Strom</a>
  </li>
</sec:authorize>

<sec:authorize access="hasPermission(#user, 'html/meters-table/show-counter')">
  <th>Zählerstand</th>
  <th>Register</th>
</sec:authorize>
```

Abbildung 11: Abfrage der Benutzergruppe im Template

Der Bereich innerhalb des *authorize*-Tags wird nur ausgegeben, wenn der angemeldete Benutzer die Benutzergruppe Admin besitzt. Somit lässt sich die Ausgabe in Abhängigkeit von der Benutzergruppe steuern. Die Benutzergruppe muss natürlich serverseitig bei den einzelnen Funktionen noch einmal geprüft werden, um sicherzugehen, dass der Benutzer die entsprechende Funktion beziehungsweise URL aufrufen darf.

### 4.2.4 Visualisierung der Meter-Daten mithilfe von Highcharts

Für die Visualisierung der Meter-Daten wird das JavaScript-Frameworks Highcharts verwendet. Highcharts kennt bereits fertige Diagrammtypen und bietet Flexibilität, einfache Integration und ist gut dokumentiert.

Prinzipiell lassen sich die JavaScript-Dateien, die für die Visualisierung zuständig sind, in drei verschiedenen Kategorien einteilen:

- Model: SmartvisChartsModel, SmartvisDataSource
- Controller: SmartvisCharts
- Configuration

In der Klasse *SmartvisChartsModel* werden die einzelnen Pfadsegmente für die URL definiert. Da der Benutzer die Möglichkeit besitzen soll, verschiedene Zeiträume, Auflösungen und Daten wie zum Beispiel Verbrauch, Spannung, Strom oder Frequenz für die Visualisierung auszuwählen, sind diese Parameter in dieser Klasse definiert. Dies hat den Vorteil, dass man sie später einfach austauschen kann, falls sich etwas ändern sollte.

Die Klasse *SmartvisDataSource* ist eine Model-Klasse, die für das Auslesen der Daten und das Zusammensetzen der URL zuständig ist. Mit dieser Klasse kann man verschiedene Datenquellen mit den in der Klasse *SmartvisChartsModel* vorhandenen Parametern definieren, die man später dem Controller *SmartvisCharts* übergibt, um so verschiedene Daten in den Diagrammen darzustellen. Somit erhält man eine gute Trennung zwischen der Controllerschicht und der Datenschicht. Die Meter-Daten werden dabei mit einem Ajax-Request abgerufen, indem man die Funktion *loadData* in der *SmartvisDataSource*-Klasse aufruft. Da die Meter-Daten im JSON-Format geliefert werden, genügt der Aufruf der jQuery-Funktion *getJSON* um die Meter-Daten abzurufen, wie in Abbildung 12 gezeigt.

```

/**
 * get data from the url
 *
 * @param function callback
 * @return void
 */
this.loadData = function(callback) {
    $.getJSON(getUrl(), callback);
}

```

Abbildung 12: Funktion loadData zum Abrufen der Meter-Daten

Der Server-Response kann direkt in ein JavaScript-Objekt umgewandelt werden, wie in Abbildung 13 gezeigt. Die Schnittstellenbeschreibung für den Aufbau der URL ist in Abschnitt 3 beschrieben.

GET http://localhost:8080/smartvis/0/meterdata/1/volt...02T00:00:00.000Z/2015-01-31T00:00:00.000Z

Header	Antwort	JSON	Cookies
Nach Schlüssel sortieren			
+	0	[ "2014-12-02T00:59:50", 237.55865921787694 ]	
+	1	[ "2014-12-02T01:59:50", 238.2875346260388 ]	
+	2	[ "2014-12-02T02:59:50", 238.79432432432438 ]	
+	3	[ "2014-12-02T03:59:50", 238.96351851851827 ]	
+	4	[ "2014-12-02T04:59:50", 237.34842592592616 ]	
+	5	[ "2014-12-02T05:59:50", 236.17435185185195 ]	
+	6	[ "2014-12-02T06:59:50", 236.24685185185203 ]	

Abbildung 13: JSON-Daten von der Schnittstelle

Die Instanziierung eines neuen *SmartvisDataSource*-Objektes wird in Abbildung 14 gezeigt.

```

var dataSource1 = new SmartvisDataSource({
    name: 'Strom [mA]',
    segment: '/smartvis/0/meterdata',
    resolution: SmartvisChartsModel.dataResolution.HOUR,
    channel: SmartvisChartsModel.dataChannel.POWER,
    interval: 1,
    from: start,
    to: end,
    yAxis: 0,
    color: '#129490'
});

```

Abbildung 14: Instanziierung eines neuen SmartvisDataSource-Objektes

Die Klasse *SmartvisCharts* bildet das Kernstück der Visualisierung. Mithilfe dieser Klasse lassen sich Diagramme erstellen und auf der Webseite anzeigen. Diese Klasse übernimmt auch die Highcharts-Konfiguration und Instanziierung. Dafür wird ein HTML-Element wie zum Beispiel ein Div-Container benötigt. Über die Funktion *createChart* wird ein neues Diagramm erstellt. Als Parameter benötigt man einen Selector auf das gewünschte HTML-Element, in dem das Diagramm angezeigt werden soll, ein Array aus *SmartvisDataSource*-Objekten und ein Objekt für weitere Optionen, wie in Abbildung 15 gezeigt.

```
var smartVisCharts = new SmartvisCharts();  
var myChart = new smartVisCharts.createChart('#my-chart', [  
    dataSource1, dataSource2  
], options);
```

Abbildung 15: Erstellen eines neuen Diagramms mithilfe der SmartvisCharts-Klasse

Dieses Diagramm würde auf der Webseite wie in Abbildung 16 gezeigt aussehen, wobei *dataSource1* die Datenreihe Strom und *dataSource2* die Datenreihe Spannung entspricht.

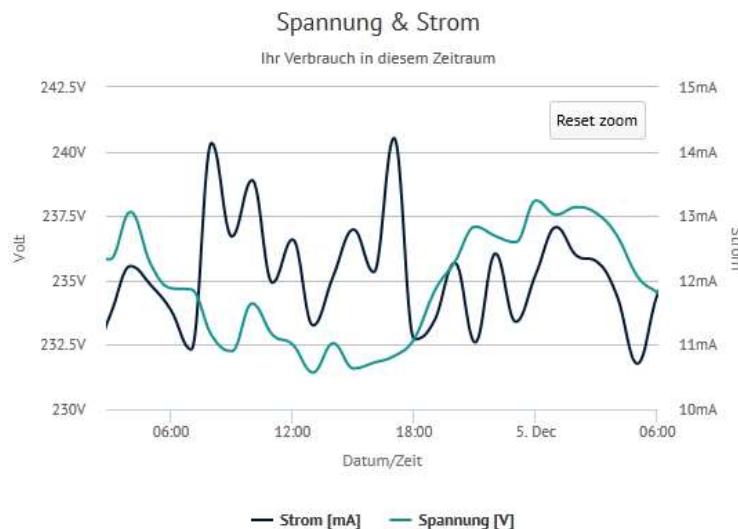


Abbildung 16: Anzeige eines Diagramms auf der Webseite

Da der Benutzer auch die Möglichkeit besitzt, die Parameter Zeitraum, Auflösung und Intervall nachträglich zu bestimmen, gibt es die Funktion *smartvisReload*, mit der die Meter-Daten mit den geänderten Parametern geladen und angezeigt werden können. Die Parameter müssen nur der Funktion übergeben werden, danach werden die neuen Daten mittels Ajax nachgeladen und angezeigt. Dafür wird auf der Webseite ein Formular mit den Einstellmöglichkeiten unterhalb des Diagramms angezeigt. Drückt der Benutzer auf den Button *Anwenden*, wird das Diagramm aktualisiert, indem die vorhin beschriebene Funktion *smartvisReload* aufgerufen wird. Die Parameter werden dabei direkt aus dem Formular ausgelesen.

### Einstellungen:

Von

Auflösung

Bis

Intervall

Abbildung 17: Einstellmöglichkeiten für die Diagramme

Abbildung 18 zeigt den Aufruf der Funktion *smartvisReload* mit den Parametern aus dem Formular aus Abbildung 17.

```
myChart.smartvisReload({  
  from: $('input[name="from"]', context).pickadate('picker').get('select').obj,  
  to: $('input[name="to"]', context).pickadate('picker').get('select').obj,  
  resolution: $('select[name="resolution"]', context).val(),  
  interval: $('select[name="interval"]', context).val()  
});
```

Abbildung 18: Aufruf der Funktion *smartvisReload*

### 4.3 Controller

In diesem Unterabschnitt werden die mit Spring MVC umgesetzten Controller für die SmartVis Implementierung beschrieben.

Grundsätzlich wird eine Klasse, welche den Aufgabenbereich des Controllers übernehmen soll, durch die Annotation *Controller* gekennzeichnet, damit Spring MVC diese dem Applikationskontext hinzufügen und die Verwaltung für diese Klasse übernehmen kann.

In Spring MVC werden alle Anfragen an die Webapplikation zentral über das Dispatcher-Servlet verwaltet und an den richtigen Controller weitervermittelt. Diese Zuordnung von HTTP-Anfrage und zugehöriger Methode eines Controllers passiert über die Annotation *RequestMapping*. Dieser Annotation wird mindestens mittels des Attributs *value* eine URI mitgegeben, da diese den wesentlichen Schlüssel in der Anfrage-Controller-Zuordnung darstellt. Hier kann auf Attribute, welche über die URI mitgegeben werden, über die Annotation *PathVariable* im Methodenkopf als Attribut zugegriffen werden. Zudem kann auch die Anfragemethode spezifiziert werden, um weiter differenzieren zu können, sowie der Rückgabotyp festgelegt werden (siehe Listing 9).

```
@RequestMapping(value = {"/{dbSource}/metermetadata"})  
public @ResponseBody  
List<MeterMetaData> getAllMeterMetaData(@PathVariable() int dbSource) {  
  return meterMetaDataService.findAll(dbSource);  
}
```

Listing 9: Beispiel einer Controllermethode

Danach folgt die eigentliche Methode. Hier kann man in SmartVis zwei Varianten unterscheiden: datenspezifische Methoden und View spezifische Methoden.

- Datenspezifische Methoden haben gemein, dass sie Daten des Gateway oder des Datacenter zurückgeben, seien es Meter- oder Metadaten. Sie sind gekennzeichnet durch die zusätzliche Annotation *ResponseBody*. Sie geben als Rückgabewert eine Liste von Objekten zurück, welche durch die Bibliothek Jackson in das JSON-Format gewandelt wird. Sie sind also nicht an eine Visualisierung durch eine bestimmte View gebunden, sondern werden clientseitig per JavaScript verarbeitet. Das Einbinden dieser Bibliothek als Abhängigkeit im POM-File reicht dabei aus. Das Attribut *runtime* zeigt Maven dabei an, dass diese Bibliothek zur Laufzeit der

Applikation zur Verfügung stehen soll (siehe Listing 10). Somit kann Spring, welches den Klassenpfad scannt, Jackson zum Einsatz bringen.

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.4.3</version>
  <scope>runtime</scope>
</dependency>
```

*Listing 10: Maven Dependency-Angabe zur Jackson-Bibliothek*

- Viewspezifische Methoden leiten entweder direkt zu einer View weiter (siehe Listing 11) in dem sie einfach gleich den Viewnamen zurückgeben, oder sie geben zusätzlich zum Viewnamen auch noch Objekte, als Model bezeichnet, an den ViewResolver weiter. Dies wird über ein sogenanntes ModelAndView Objekt erledigt, ein Spring spezifisches Objekt. Diesem werden als Model jegliche Javaobjekte als auch ein Bezeichner für jedes Objekt mitgegeben, damit in

```
@RequestMapping(value = {"/","index"})
public String index()
{
    return "index";
}
```

*Listing 11: Beispiel einer viewspezifischen Controllermethode*

In der SmartVis Applikation ist auch die Möglichkeit eingebaut, über einen Einstellungsbereich die Datenbankverbindung zum Gateway oder Datacenter dynamisch, während der Webserver läuft, zu ändern. Dabei wird auch eine Validierung der, durch den Benutzer eingegebenen und für die Datenbankverbindung relevanten Daten, vorgenommen. Dem entsprechend werden die an den Controller zurückgegebenen Einstellungsdaten erst an eine Validierungsmethode übergeben. Hier wird überprüft, ob sich die Datenbank mit den angegebenen Einstellungen über JDBC erreichen lässt. Ist das der Fall, werden die Datenbankeinstellungen, welche in einem Properties-File hinterlegt sind, durch die gerade validierten überschrieben und der Applikationskontext von Spring wird neu geladen. Dadurch werden die Konfigurationsklassen neu geladen und somit Spring mit den aktuellen Einstellungen initiiert. Außerdem wird der Nutzer darauf hingewiesen, dass sehr wohl die Verbindung korrekt ist, jedoch seitens der Applikation nicht das eingesetzte Datenbankschema verifiziert werden kann. Kann keine Verbindung aufgebaut werden, werden die alten Einstellungen beibehalten und eine Fehlermeldung wird zurückgegeben.

## 5 Sicherheitsaspekte

Als Web-Applikation muss SmartVis ein Mindestmaß an Sicherheitsfunktionen bieten. Folgender Abschnitt erläutert die implementierten Aspekte im Detail.

### 5.1 Authentifizierung und Autorisierung (engl. AAA)

Zu den wichtigsten Sicherheits-Mechanismen einer Webseite zählen der Schutz gegen unerlaubte und unautorisierte Zugriffe auf benutzerspezifische Daten. In der Praxis bedeutet dies, dass sich ein Benutzer identifizieren muss, bevor dieser Zugriff auf die nicht öffentlich einsehbaren Bereiche einer Webseite bekommt. Bei der Benutzer-Authentifizierung in der Form eines Login-Fensters werden üblicherweise nur der Benutzername und das zugehörige Passwort abgefragt. Die Eingabe der Daten ermöglicht jedoch nur eine Identifizierung des Benutzers und sagt nichts darüber aus, welche Aktionen dieser auf der Webseite ausführen darf. Aus diesem Grund müssen den Benutzern gewisse Berechtigung zugewiesen werden. Diese werden üblicherweise nicht einzelnen Benutzern, sondern ganzen Benutzergruppen zugeordnet. Da verschiedene Benutzergruppen meist unterschiedliche Rollen wahrnehmen wird die Verknüpfung von Benutzern und deren Rollen auch rollenbasierte Berechtigungskonzept bezeichnet. Es bietet die Möglichkeit einer Autorisierung bestimmter Benutzergruppen für den gesicherten Zugriff auf Daten oder Funktionen von Web-Applikationen.

### 5.2 Berechtigungskonzept

SmartVis verfügt über ein flexibel erweiterbares rollenbasiertes Berechtigungskonzept. Einerseits können jedem Benutzer eine oder mehrere Rollen zugeordnet werden, andererseits können diese Rollen wiederum unterschiedliche Berechtigungen aufweisen. Auf Grund seiner einfachen Implementierung können sowohl die Rollen als auch einzelnen Berechtigungen zur Benutzer-Autorisierung herangezogen werden. Zusätzlich erlaubt das Konzept einer „Deny-All-Strategy“ eine Zusammenführung aller Berechtigungen aus den verschiedensten Rollen. Daraus lassen sich weitere Möglichkeiten zur Autorisierung ableiten.

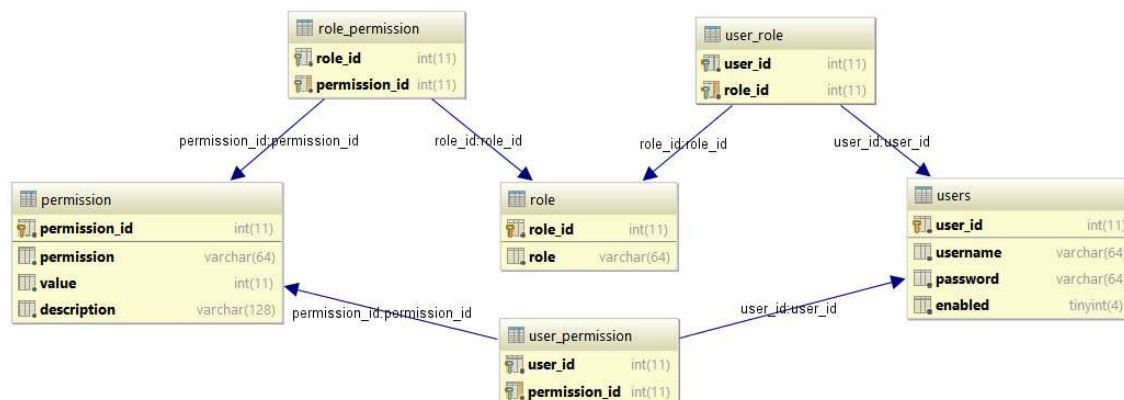


Abbildung 19 – ER-Modell des Berechtigungskonzepts

Abbildung 19 veranschaulicht das ER-Modell des Berechtigungskonzepts im Detail. Es ist zu erkennen, dass zwischen den Tabellen Berechtigungen (engl. permissions), Rollen (engl. roles) und den Benutzern (engl. user) jeweils eine Zuordnungstabelle eingefügt wurde. Somit lassen sich keine oder mehrere Berechtigungen zu Rollen zuordnen und keine oder mehr Rollen zu einem Benutzer zuordnen. Da diese Methode es jedoch nicht zuließ einem Benutzer zusätzliche Berechtigung einzuräumen, wurde auch eine Zuordnungstabelle zw. Berechtigungen und Benutzer realisiert.



### 5.2.1 Installation

Die Installation des Berechtigungskonzepts erfolgt mittels SQL-Skript<sup>1</sup>, welches alle Tabellen und Einträge erzeugt um sich auf der SmartVis-Applikation einzuloggen. Die Installation erfolgt in einer gesonderten SmartVis-Datenbank, welche komplett getrennt von der Meterdaten-Datenbank gehalten wird. **Es ist darauf zu achten, dass alle Default-Zugänge deaktiviert werden**, sobald das System in einer Kundenumgebung läuft. Die Passwörter selbst sind in der DB mittels Hashes<sup>2</sup> abgelegt.

### 5.2.2 Rollen

Das rollenbasierte Berechtigungskonzept der SmartVis-Applikation verfügt über ein erweiterbares Rollenmanagement, welche es erlaubt beliebig viele Rollen hinzuzufügen bzw. zu entfernen. Bei der Installation wird für jede Rolle ein Default-Benutzer angelegt. Diese sind in der folgenden Tabelle 4 erläutert.

	Beschreibung	Benutzer	Passwort
<b>ROLE_ADMIN</b>	Administrator od. Techniker	admin	admin
<b>ROLE_USER</b>	Kunde od. normaler Benutzer	kunde	kunde
<b>ROLE_DSO</b>	Distributed System Operator	dso	dso
<b>ROLE_DC</b>	Datacenter	dc	dc
<b>ROLE_EPS</b>	Electronic Provider Service	eps	eps

Tabelle 4 - Rollen und Default-Benutzer

### 5.2.3 Berechtigungen

Das Gros der Berechtigung der SmartVis-Applikation können in zwei Kategorien unterteilt werden. Einerseits werden Rechte benötigt um der Anforderung gerecht zu werden, dass Benutzergruppen unterschiedliche UI-Elemente in HTML-Webseite aufweisen müssen. Dies ermöglicht beispielsweise eine vereinfachte Darstellung der Informationen für die Benutzergruppe „Kunde“. Andererseits muss der Zugriff auf die REST-Schnittstelle ebenso mit gewissen Berechtigungen abgesichert werden. Das Resultat ist damit eine gehärtete Daten-Schnittstelle, welche es nur autorisierten Benutzergruppen ermöglicht auf die für Sie relevanten Informationen lesend oder schreibend zuzugreifen. Damit sich diese beiden Kategorien bereits bei der Beschreibung deutlich unterscheiden, wurde ein Namenschema für Benennung von Berechtigungen entwickelt.

	Beschreibung	Beispiele
<b>HTML</b>	Berechtigungen die die Visualisierung betreffen beginnen mit „ <b>html/</b> “. Darauf folgt ein Pfad und abschließend der Name der Berechtigung. Als Präfix wird „ <b>show</b> “ verwendet.	html/navigation/show-dashboard html/events/show-logs
<b>REST</b>	Berechtigungen die die Rest-Schnittstelle betreffen beginnen mit „ <b>rest/</b> “. Darauf folgt der Pfad und der Name der Berechtigung.	rest/meter-data/get-meters-ignore-users rest/events/get-errors
<b>AUTH</b>	Berechtigungen die die Einschränkung der Datenauswahl betreffen beginnen mit „ <b>auth</b> “.	auth/is-admin auth/is-customer
<b>{DS_n}</b>	Berechtigungen, die die Daten selbst betreffen. Die variablen Werte werden in geschweiften Klammern angegeben	DS_1/customer/2 DS_2/customer/1

Tabelle 5 - Gegenüberstellung der Berechtigungskategorien

<sup>1</sup> Es handelt sich um einen Teil des SmartVis-Datenmodells, welches zum Projekt abgelegt ist.

<sup>2</sup> Die Erzeugung und die Überprüfung der Hashes erfolgt durch den Spring Security BcryptPasswordEncoder.

### 5.2.4 Berechtigungen zu Rollen Zuordnung

Das Berechtigungskonzept der SmartVis-Applikation ermöglicht es verschiedenen Benutzern mehr als eine Rolle zuzuordnen. Meist wird dieser jedoch eine stimmte Rolle einnehmen. Die Rollen waren aus den Anforderungsdokumenten abzuleiten und wurden mit folgende Berechtigungen verknüpft.

Berechtigung	Techniker (Admin)	Kunde (User)	Distributed System Operator	Data Center	Electric Power Supplier
html/navigation/show-dashboard	•	•			
html/navigation/show-meter-data	•	•	•	•	•
html/navigation/show-settings	•	•	•	•	•
html/navigation/show-users	•				
html/navigation/show-events	•	•	•	•	•
html/meter-data/show-voltage	•		•	•	•
html/meter-data/show-current	•	•			•
html/meter-data/show-current-phases	•				•
html/meter-data/show-consumption	•	•			•
html/meter-data/show-consumption-phases	•				•
html/meter-data/show-power	•	•			•
html/meter-data/show-power-phases	•				•
html/meter-data/show-frequency	•		•	•	•
html/meters-table/show-id	•				
html/meters-table/show-counter	•	•			
rest/events/get-admin-error	•		•		
rest/events/get-admin-log	•		•		
rest/events/get-customer-info	•	•	•		
html/live-data/show-voltage	•		•	•	•
html/live-data/show-consumption	•				•
html/live-data/show-consumption-phases	•				•
html/live-data/show-power	•	•			•
html/live-data/show-power-phases	•				•
html/live-data/show-frequency	•		•	•	•
html/dashboard/default-resolution-hour	•	•			
html/dashboard/default-resolution-day					
html/dashboard/default-update-10					
html/dashboard/default-update-30					
html/dashboard/default-update-60	•	•			
auth/is-customer		•			
auth/is-admin	•		•	•	•

Tabelle 6 - Standardmäßig eingestellte Berechtigungen zu Rollen Zuordnung



### 5.2.5 Berechtigungen zu User Zuordnung

Für jeden Anwender muss hinterlegt werden, auf welche Kundendaten dieser zugreifen darf.

Das String-Format dieser Berechtigung hat folgenden Aufbau:

```
{datasource_id}/customer/{customer_id}
```

{datasource\_id} - gibt die Datenquelle/Datenbank an (siehe 6.2)

{customer\_id} - gibt den Kunden innerhalb der Datenbank an (customer\_id in der Tabelle customers).

Bei Auslieferung werden die Berechtigungen auf Datenbank1 mit Kunde 1 und 2 hinterlegt.

Diese Berechtigungsstrings sind in der Datenbank db\_smartvis.permissions manuell zu pflegen und sind Voraussetzung für die Berechtigungszuordnung über die SmartVis-Benutzeroberfläche.

## 6 Einstellungen SmartVis

### 6.1 SmartVis-Anwendungseinstellung

Diverse Einstellungen der SmartVis-Anwendung können in der Datei „application.properties“ vorgenommen werden:

Es kann eingestellt werden, welche Methode zum Ermitteln der Smart-Meter-Werte angewendet werden soll (siehe Listing 12):

```
#Mode to retrieve/calculate Meter-Values
# 0 = get all values from DB and perform interpolation<br/>
# 1 = get intervall-borders from DB and calculate interpolated values from counter<br/>
# 2 = get aggregated values from DB (WITHOUT interpolation)
smartvis.meterValueRetrievalMode=2
```

Listing 12 – application.properties – meterValueRetrievalMode

Einstellungen zu H0-Profilen:

```
# H0-Profil:
# das H0-Profil enthaelt Viertelstunden-Leistungswerte für einen Jahresverbrauch von 1.000
kWh/a
# mit der u.a. Integer-Einstellung werden die hinterlegten H0-Werte aliquotiert
# Beispielwerte finden sich hier:
# http://stromliste.at/nuetzliche-infos/durchschnittlicher-stromverbrauch#stromverbrauch-1-
personen-haushalte
smartvis.profileReferenceValueKwYear=2000

# Pseudo-Datenbankquelle für H0-Datenbank
smartvis.dbSourceH0=100
```

Listing 13 – application-properties – Einstellung zu H0-Profilen

Einstellungen zum Logging:

```
#
logging.level.org.springframework.web=INFO
logging.level.at.ac.fhsalzburg.smartvis=INFO
```

LOGGING

Listing 14 – application-properties – Logging-Einstellungen

Einstellen des http-Server-Port:

```
#
server.port=8080
```

Port

for

embedded

HTTP-Server

Listing 15 – application-properties –HTTP- Server-Port

## 6.2 Datenbank konfigurieren

Die SmartVis-Anwendung benötigt zumindest zwei (logisch getrennte) Datenbanken:

- eine Datenbank für SmartVis-Daten enthält z.B. Benutzer- und Rechte-Informationen (siehe Abbildung 20 - db\_smartvis)
- eine oder mehrere Datenbanken mit Smart-Metering-Daten, z.B. Messdaten und Meta-Daten von Smart-Meter-Geräten (siehe Abbildung 20 - db\_meters1 und db\_meters2)

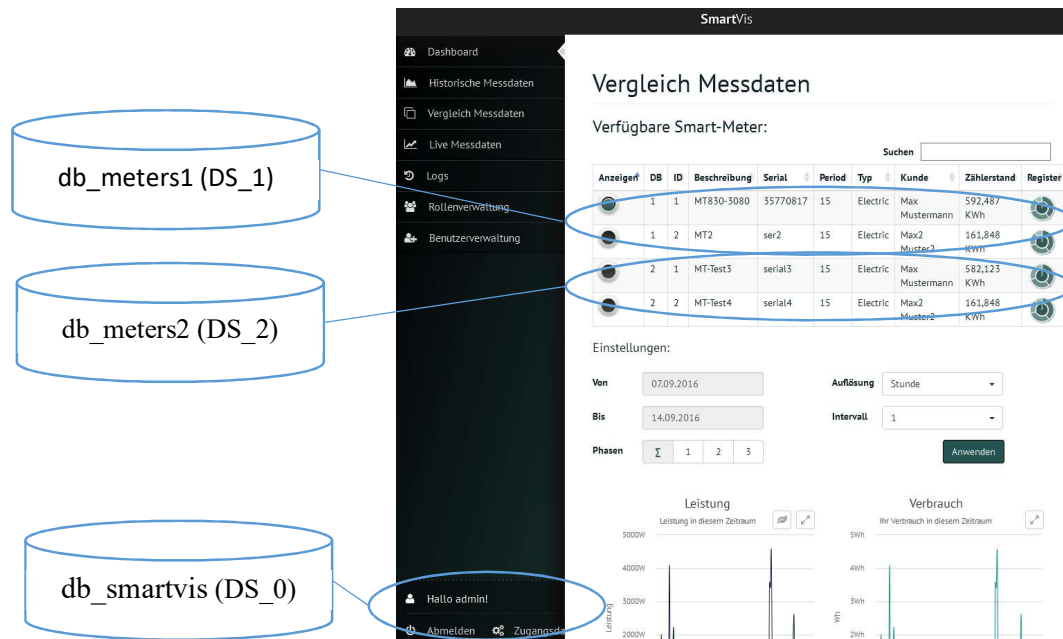


Abbildung 20 – Datenbanken

In der SmartVis-Konfigurationsdatei ../WEB-INF/classes/db/database-config.xml müssen die DB-Anbindungen eingetragen sein (siehe Listing 16):

<pre> &lt;bean id="dataSource" primary="true"       class="at.ac.fhsalzburg.smartvis.common.DbRoutingDataSource"&gt;   &lt;property name="targetDataSources"&gt;     &lt;map key-type="java.lang.String"&gt;       &lt;entry key="DS_0" value-ref="dataSourceDbSmartVis"/&gt;       &lt;entry key="DS_1" value-ref="dataSourceDbMeter1"/&gt;       &lt;entry key="DS_2" value-ref="dataSourceDbMeter2"/&gt;     &lt;/map&gt;   &lt;/property&gt;   &lt;property name="defaultTargetDataSource" ref="dataSourceDbSmartVis"/&gt; &lt;/bean&gt; </pre>	(1)
<pre> &lt;bean id="parentDataSource"       class="at.ac.fhsalzburg.smartvis.common.DriverManagerDataSource"&gt;   &lt;property name="driverClassName" value="com.mysql.jdbc.Driver" /&gt;   &lt;property name="url" value="jdbc:mysql://localhost:3306/db_meters"/&gt;   &lt;property name="username" value="root" /&gt;   &lt;property name="password" value="root" /&gt; &lt;/bean&gt; </pre>	(2)
<pre> &lt;bean id="dataSourceDbSmartVis" parent="parentDataSource" &gt;   &lt;property name="url" value="jdbc:mysql://localhost:3306/db_smartvis"/&gt; &lt;/bean&gt; </pre>	(3)
<pre> &lt;bean id="dataSourceDbMeter1" parent="parentDataSource" /&gt; </pre>	(4)
<pre> &lt;bean id="dataSourceDbMeter2" parent="parentDataSource" &gt; </pre>	(5)

```
<property name="openForAccess" value="true" />
</bean>
```

Listing 16 - DB-Attribute in database-config.xml

- (1) Die Bean „dataSource“ ist eine „AbstractRoutingDataSource“ des Spring-Frameworks. Sie ermöglicht die dynamische Steuerung der Datenbank-Zugriffe auf die definierten Zieldatenbanken („targetDataSources“). Die referenzierten Datenbanken („value-ref“) müssen als „DataSource“-Beans deklariert werden (siehe Pkt 2 – 5).
- (2) Die abstrakte Bean „parentDataSource“ definiert allgemeine DB-Attribute, die von den „Child“-Beans übernommen werden, sofern diese keine abweichenden Definitionen enthalten.
- (3) Die Bean „dataSourceDbSmartVis“ definiert die Datenbank für SmartVis (db\_smartvis)
- (4) + (5) + ... Die weiteren Beans („dataSourceDbMeter1“ + ...) sind Datenbank-Definitionen für Smart-Metering-Daten (db\_meters). Die Zahl entspricht der „dbConnection“ der Rest-Schnittstelle (siehe 3 Konzeption der Webschnittstelle). Mit der Einstellung „openForAccess“ werden Datenbanken gekennzeichnet, die für alle Anwender ohne Prüfung von Zugriffsrechten verfügbar sind.

Wenn die Einstellungen nicht bereits vor der Erstellung des War-File erfolgt ist, kann die Datei auch im gepackten War-File (z.B. mit dem Tool 7-Zip) abgeändert werden.

### 6.3 H0-Profil - Datenbasis

Für vergleichende Darstellungen bietet SmartVis die Möglichkeit, H0-Profildaten einzublenden. Diese H0-Daten werden bei Anwendungsstart aus dem Excel-File

./resources/VDEW\_Standardlast\_Profile\_Strom\_VSG.xlsx

geladen.

	A	B	C	D	E	F	G	H	I	J
1	<b>Standardlastprofil H0 - Haushalt</b>									
2	<b>H0</b>	<b>Winter</b> (01.11. - 20.03.)			<b>Sommer</b> (15.05. - 14.09.)			<b>Übergangszeit</b> (21.03. - 14.05. und 15.09. - 31.10.)		
3	[kW]	Samstag	Sonntag	Werktag	Samstag	Sonntag	Werktag	Samstag	Sonntag	Werktag
5	0:30	0,0682	0,0811	0,0608	0,0849	0,0925	0,0769	0,0751	0,0868	0,0696
6	0:45	0,0659	0,0750	0,0549	0,0807	0,0859	0,0688	0,0707	0,0812	0,0624
7	1:00	0,0633	0,0691	0,0499	0,0766	0,0799	0,0624	0,0666	0,0757	0,0566
8	1:15	0,0595	0,0634	0,0462	0,0717	0,0741	0,0580	0,0623	0,0701	0,0525
9	1:30	0,0550	0,0582	0,0436	0,0666	0,0687	0,0553	0,0580	0,0645	0,0497
10	1:45	0,0505	0,0536	0,0419	0,0616	0,0639	0,0536	0,0541	0,0593	0,0479
11	2:00	0,0466	0,0499	0,0408	0,0574	0,0599	0,0524	0,0508	0,0549	0,0466
12	2:15	0,0439	0,0473	0,0401	0,0545	0,0570	0,0513	0,0484	0,0517	0,0455
13	2:30	0,0402	0,0455	0,0396	0,0508	0,0550	0,0502	0,0468	0,0494	0,0445

Abbildung 21 - Standardlastprofil H0 - Haushalt

Bei Bedarf können diese H0-Daten inhaltlich aktualisiert werden. Das Format des ersten Tabellenblatts H0 darf sich jedoch nicht verändern, da die Einspiellogeik darauf abgestimmt ist.

### 6.4 Benutzer- und Rollenverwaltung

Die Benutzer- und Rollenverwaltung ist Teil der Anwendungseinstellungen und wird in 5.2 beschrieben. In der SmartVis-Anwendung ist für berechnete Anwender eine Benutzer- und Rollenverwaltung vorgesehen. Die Abbildung 22 Rollen- und Benutzerverwaltung in der SmartVis Applikation Abbildung 22 zeigt, wo diese Einstellungen getroffen werden können.

## Rollenverwaltung







Rollen:

 Rolle anlegen

10

Einträge anzeigen

Suchen

Beschreibung	Aktion
Administrator	 Verwalten
Datacenter Operator	 Verwalten
Datacenter User	 Verwalten
Electric Power Service	 Verwalten
Kunde	 Verwalten
ShowView	 Verwalten

1 bis 6 von 6 Einträgen

## Benutzerverwaltung







Benutzer:

 Benutzer anlegen

10

Einträge anzeigen

Suchen

Benutzername	Aktion
admin	 Verwalten
dc	 Verwalten
dso	 Verwalten
eps	 Verwalten
kunde	 Verwalten
show	 Verwalten

1 bis 6 von 6 Einträgen

Abbildung 22 Rollen- und Benutzerverwaltung in der SmartVis Applikation

Die Daten zu Benutzern und dessen Berechtigungen sind in der SmartVis-Datenbank hinterlegt. Die Berechtigung zur Administration von Benutzern lautet „**html/navigation/show-users**“ mit der **permission\_id 4**. Sollte man sich versehentlich vom Zugang zur UI Benutzerverwaltung aussperren, so muss man für eine administrative Rolle die Berechtigung in die Tabelle „role\_permissions“ eintragen.

Allgemein sind in der Datenbank SmartVis in der Tabelle „permissions“ alle Berechtigungen auch textlich beschrieben. Die Berechtigungen „auth/is-customer“ und „auth/is-admin“ werden hier auch noch einmal beschrieben.

Ersterer erlaubt eine dynamische Auswahl der Messdaten, welche spezifisch für die Kunden zugeschnitten sind. Die Konfiguration selbst erfolgt Customer.JSON Ressource-Datei, welche die Parameter für Einstellung der Messdaten beschränkt.

Zweiterer wird – wie der Name sagt – ausschließlich für Administratoren benötigt. Diese ermöglicht die Abfrage und Anzeige von Smart-Meter-Daten, welche nicht auf den aktuellen Benutzer eingeschränkt sind bzw. welche den Zugriff auf alle Datenbanken ermöglicht.

## 7 Installation SmartVis auf Raspberry Pi

In diesem Abschnitt werden die Installationsschritte von SmartVis auf einem Raspberry Pi beschrieben.

### 7.1 Voraussetzungen

Folgende Komponenten werden für die Installation von SmartVis auf einem Raspberry 2 vorausgesetzt:

- Java Runtime Environment (ab Version 1.8)
- MySQL-Datenbank (ab Version 5.1) mit Gateway/Datacenter-Schema
- IP-Anbindung

### 7.2 SmartVis-Datenbank anlegen

SmartVis-spezifische Daten (z.B. für Berechtigungen, Rollen, ...) werden in einer eigenen Datenbank geführt. Ein ausführbares SQL-Skript zur Erstellung der benötigten Tabellen und der initialen Benutzer, Rollen und Berechtigungen ist im SmartVis-Anwendungspaket enthalten:

```
..\WEB-INF\classes\db\create.db_smartvis.sql
```

Um auf diese Datei (im gepackten War-File) zugreifen zu können, ist diese mit einem Tool (z.B. 7-Zip) zu öffnen.

Zur Konfiguration des Datenbankzugriffs wird auf Kapitel 6.2 verwiesen.

### 7.3 SmartVis-Anwendung auf Raspberry Pi installieren (Deployment)

Die SmartVis-Anwendung liegt in Form einer Web-Archive-Datei (.war-File) vor und enthält einen „embedded“ Tomcat-Server. Die Java-Anwendung SmartVis kann somit mit dem Befehl `java -jar` ausgeführt werden. Dabei startet die Web-Anwendung auf dem konfigurierten Port (siehe 6.1).

Beispielhaft wird in dieser Anleitung ein SmartVis-Verzeichnis in `/etc` erstellt:

```
mkdir /etc/smartvis
```

Grundsätzlich kann das Web-Archive-File in jedem beliebigen Verzeichnis abgelegt und von dort gestartet werden. Das eigentliche Deployment besteht aus der Übertragung des Web-Archive-File (`smartvis-1.0-SNAPSHOT.war`) in das SmartVis-Verzeichnis (z.B. mit WinSCP).

### 7.4 SmartVis-Anwendung auf Raspberry Pi automatisch starten

Damit die SmartVis-Anwendung einfacher in den Prozess-Listen zu identifizieren ist, wird ein Symlink-Befehl `java-smartvis` für das Starten der SmartVis-Anwendung erzeugt:

```
ln -s /usr/bin/java /etc/smartvis/java-smartvis
```

Start-Skript `smartvis-start.sh`:

```
#!/bin/bash
/etc/smartvis/java-smartvis -jar /etc/smartvis/smartvis-1.0-SNAPSHOT.war
```

Stop-Skript `smartvis-stop.sh`:

```
#!/bin/bash
# Grabs and kill a process from the pidlist that has the word smartvis

pid=`ps aux | grep smartvis | awk '{print $2}'`
kill -9 $pid
```

Im nächsten Schritt werden die Start- und Stop-Skripte als Service registriert. Dazu wird das Skript `/etc/init.d/smartvis` erstellt:

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:          smartvis
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start/Stop SmartVis
# Description:       Service-Script fuer SmartVis-Webanwendung
### END INIT INFO
#
case $1 in
  start)
    /bin/bash /etc/smartvis/smartvis-start.sh
    ;;
  stop)
    /bin/bash /etc/smartvis/smartvis-stop.sh
    ;;
  restart)
    /bin/bash /etc/smartvis/smartvis-stop.sh
    /bin/bash /etc/smartvis/smartvis-start.sh
    ;;
  *)
    ;;
esac
exit 0
```

Die Skripte müssen ausführbar sein, deshalb:

```
root@jrz-smartvis:/etc/smartvis# chmod +x smartvis-stop.sh
root@jrz-smartvis:/etc/smartvis# chmod +x smartvis-start.sh
root@jrz-smartvis:/etc/smartvis# chmod +x /etc/init.d/smartvis
```

Mit `service smartvis start` kann die SmartVis-Anwendung nun in einer Shell gestartet werden, mit `service smartvis stop` wird die SmartVis-Anwendung beendet.

Das automatische Starten von Services erfolgt über Symlink-Einträge in `rc#.d`-Verzeichnissen. Der folgende Befehl erzeugt die benötigten Einträge für SmartVis:

```
update-rc.d smartvis defaults
```

Ergebnis:

```
root@jrz-smartvis:/# ls /etc/rc*/*smart* -l
lrwxrwxrwx 1 root root 18 Dec  8 08:06 /etc/rc0.d/K01smartvis -> ../init.d/smartvis
lrwxrwxrwx 1 root root 18 Dec  8 08:06 /etc/rc1.d/K01smartvis -> ../init.d/smartvis
lrwxrwxrwx 1 root root 18 Dec  8 08:06 /etc/rc2.d/S03smartvis -> ../init.d/smartvis
lrwxrwxrwx 1 root root 18 Dec  8 08:06 /etc/rc3.d/S03smartvis -> ../init.d/smartvis
lrwxrwxrwx 1 root root 18 Dec  8 08:06 /etc/rc4.d/S03smartvis -> ../init.d/smartvis
lrwxrwxrwx 1 root root 18 Dec  8 08:06 /etc/rc5.d/S03smartvis -> ../init.d/smartvis
lrwxrwxrwx 1 root root 18 Dec  8 08:06 /etc/rc6.d/K01smartvis -> ../init.d/smartvis
```

Das Starten der SmartVis-Anwendung auf dem Pi2 dauert ca. 2:30 Minuten.

Mit dem Befehlen „`top`“ oder „`ps -A`“ kann eine Liste der laufenden Prozesse ausgegeben werden.

## 8 SmartVis Entwicklung

In diesem Abschnitt werden die Rahmenbedingungen für die SW-Entwicklung SmartVis beschrieben.

Die Quellcode-Verwaltung erfolgte mit der GitLab-Anwendung der FH-Salzburg. Das Projekt ist für alle FHS/ITS-berechtigten Anwender unter folgender Adresse erreichbar:

<https://its-git.fh-salzburg.ac.at/smartvisgroup/rd-smartvis>

An dieser Stelle wird auch auf das Kapitel 2 (Konzepte und eingesetzte Technologien) verwiesen.

### 8.1 Benötigte Komponenten:

Für die (Weiter-)Entwicklung der SmartVis-Anwendung werden folgende Komponenten benötigt:

- Java JDK ab Version 1.8
- Maven 3.x als Build-Werkzeug (ist meist auch in IDE enthalten)
- JDBC-fähige Datenbank (z.B. MySQL-Datenbank ab Version 5.x)
- optional: IDE (z.B. Eclipse, IntelliJ, ...)

### 8.2 Starten / Debuggen der Anwendung

SmartVis ist eine Spring Boot Web-Anwendung mit einem „embedded“ Tomcat-Server. Die Anwendung kann mit „java -jar <smartvis.war>“ gestartet werden (siehe <http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-running-as-a-packaged-application> ).

In einer IDE kann SmartVis als „Java Application“ (bei Bedarf auch im Debug-Modus) ausgeführt werden. Die zu startende Main-Methode befindet sich in der Klasse „SmartVisApplication“.

Sofern keine anderen Einstellungen getroffen wurden (siehe Kapitel 6.1), ist die Anwendung in der Root-Domäne mit Port 8080 aufrufbar (z.B. <http://localhost:8080>).