

Fachhochschule Bielefeld

Campus Minden

Informatik



Projektbericht in dem Modul Webengineering:

## **SmartMonitoring**

Webportal für PV-Kennlinien

Eingereicht von:

Hannes Rüffer

Michelle Vorwerk

Moritz Pit Withöft

Christiane Zolkin

Matrikelnummer:  
1151954

Matrikelnummer:  
1150973

Matrikelnummer:  
1151206

Matrikelnummer:  
1151404

Fachsemester: 5

Fachsemester: 5

Fachsemester: 5

Fachsemester: 5

Abgabedatum: 28.01.2020

Gutachter:

Prof. Dr.-Ing. Grit Behrens

Cem Basoglu  
Florian Fehring

## Inhaltsverzeichnis

1	Motivation und Themenbeschreibung.....	1
2	Vorstellung der Gruppenmitglieder .....	3
2.1	Vorstellung der Gruppenmitglieder .....	3
2.2	Aufgabenverteilung im Team.....	3
3	Theoretische Grundlagen.....	4
3.1	Photovoltaik-Zelle .....	4
3.2	MPP – Maximum Power Point .....	5
3.3	$U_L$ – Leerlaufspannung.....	6
3.4	$I_K$ – Kurzschlussstrom .....	6
3.5	Kennlinien.....	6
3.6	Hellkennlinien.....	7
3.7	Dunkelkennlinien .....	8
3.8	Nutzen des Modulportals.....	9
4	Konzeptionelle Arbeiten.....	9
4.1	Usecasediagramm .....	9
4.2	Sequenzdiagramm.....	10
4.3	Klassendiagramm .....	11
5	Code-Implementierungsdetails.....	12
5.1	Änderung von Queryparametern.....	12
5.2	Suche im Backend .....	13
5.3	Handling der Suchergebnisse .....	15
5.4	Kennlinien anzeigen .....	15
5.4.1	Kennlinien hochladen.....	18
5.5	Modulübersicht .....	19
5.6	Senden eines Kommentars.....	20
5.7	Speichern eines Kommentars im Backend .....	20
5.8	Backend Media.....	21
5.9	Backend View-Requests .....	22
5.10	Explizite Verarbeitung von Bildern.....	24
6	Ausblick und Fazit.....	24
6.1	Ausblick .....	24
6.2	Fazit .....	24
7	Installationshinweise und Benutzerhandbuch.....	25

7.1	Installationshinweise.....	25
7.2	Benutzerhandbuch.....	26
7.2.1	Modulübersicht.....	26
7.2.2	Moduldetailseite .....	26
7.2.3	Kennlinien vergleichen.....	27
7.2.4	Kennlinien hochladen.....	27
7.2.5	Fehleranalyse .....	28
7.2.6	Moduldetails downloaden .....	28
7.2.7	Module kommentieren .....	28
7.2.8	Suche nach Modulen.....	29
7.2.9	Anlegen neuer Module.....	30
8	Literaturverzeichnis.....	31
9	Anlagen.....	32

# 1 Motivation und Themenbeschreibung

Gerade in der heutigen Zeit, in der erneuerbare Energien in der Gesellschaft eine wichtige Rolle spielen, ist das Thema der Photovoltaikanlagen sowie der Solarenergie allgegenwärtiger denn je. Doch obwohl das Thema der erneuerbaren Energien immer mehr in den Fokus der breiten Öffentlichkeit rückt und Deutschland bis zum Jahr 2050 klimaneutral sein möchte<sup>1</sup>, ist im Jahre 2019 nur 43% des Stroms aus erneuerbaren Energien gewonnen worden<sup>2</sup>.

Das *Solar Computing Lab* der FH-Bielefeld am Standort Campus Minden hat es sich derweil zur Aufgabe gemacht diese Entwicklung zu beschleunigen, indem die Effizienz von Photovoltaikanlagen untersucht und aufgezeichnet werden soll, aber auch, indem beispielsweise nachhaltige Gebäudeisolierung und das Raumklima in solchen Gebäuden untersucht werden.

Durch zahlreiche Forschungsarbeiten wird dabei die Kernfrage bearbeitet, in wie fern die PV-Anlagen technisch weiterentwickelt werden können und dabei eine hohe Energieeffizienz gewährleistet werden kann. Ein besonderer Fokus liegt hierbei auch in der Einbindung der Studierenden vor Ort, indem sowohl Module im Studiengang Informatik angeboten werden, die sich im weiteren Sinne mit diesem Thema beschäftigen als auch Studierenden die Möglichkeit geboten wird, eine Abschlussarbeit im Bereich der angewandten Forschung zu schreiben<sup>3</sup>.

So wurden durch das *SCL* im Rahmen des Wahlmoduls *Webengineering* im Wintersemester 19/20 ebenfalls Studierenden in verschiedene Projekte des Solar Computing Labs eingebunden. Dabei hatten die Studierenden die Möglichkeit selbst Gruppen zu bilden und sich anschließend eines von vier möglichen Projekten auszusuchen:

- Webportal für PV-Kennlinien
- Web Applikations-Anbindung
- Progressive Webapplikation
- Anlagenvisualisierung

All diese Projekte finden im Bereich des *SmartMonitoring* statt. *SmartMonitoring* ist eine webbasierte Anwendung des SCL, in der Daten gespeichert und zur Verfügung gestellt werden. Dabei basiert ein *Smart Monitoring System* auf folgenden Kernaspekten<sup>4</sup>:

- Datenkommunikation und kryptografischen Algorithmen
- Datenspeicherung im *Monitoring System*
- *Data Mining* Applikationen

Die Gruppe, gebildet aus den Studierenden Hannes Rüffer, Michelle Vorwerk, Christiane Zolkin und Moritz Pit Withöft (im Abschnitt 2 vorgestellt) entschied sich für das Projekt, in dem ein Webportal für PV-Kennlinien angelegt werden sollte. Dieses Projekt stellt eine spannende Herausforderung, aber aus Sicht der Studierenden auch eine wichtige Rolle, da, weil eine gute Informationsquelle für PV-Module essenziell für den Verbraucher und Unternehmen ist. Durch dieses Webportal können sich Interessierte über PV-Module

---

<sup>1</sup> (BMU, 2019)

<sup>2</sup> (Diermann, 2019)

<sup>3</sup> (Lab, 2014)

<sup>4</sup> (Behrens, 2019)

und deren Leistung informieren und verschiedene Aspekte der Module miteinander vergleichen, um das beste Modul für sich selbst herauszusuchen.

Auch die anderen Projekte klangen verlockend für das Team. Doch auf Grund der breiten Reichweite an Implementierungsmöglichkeiten, sowohl im Frontend als auch im Backend, hat sich die Gruppe für dieses Projekt entschieden, um für sich selbst auch möglichst viele Erfahrungen gewinnen zu können.

Hierbei soll dem Nutzer auf der Startseite eine Übersicht mit den am meisten besuchten, den neusten und den am meisten kommentierten Modulen geboten werden, um sich einen ersten Überblick verschaffen zu können. Hier soll der Nutzer entweder die Möglichkeit haben selbst nach Modulen zu suchen, was dieser sowohl über einen Modulnamen als auch über den Herstellernamen tun kann. Andererseits kann der Nutzer auch direkt auf eine Moduldetailseite gelangen, falls er sich ein Modul auf der Übersichtsseite genauer anschauen möchte.

Auf der Moduldetailseite sollen nun genauere Informationen zum Modul angezeigt werden. Diese Informationen reichen von Hersteller über Maße des Moduls bis hin zu genaueren Eigenschaften wie der Leistung. Weiterhin werden hier die zum Modul hochgeladenen Kennlinien visuell dargestellt, verknüpft mit der Option, für angemeldete Benutzer, selbst Kennlinien hochzuladen. Außerdem werden auf der Seite noch Moduldetails zum Download angeboten, ergänzt von der Möglichkeit sich über Module auch in einer Diskussionssektion auszutauschen.

Neben der Möglichkeit Kennlinien hochzuladen, soll ein angemeldeter Nutzer natürlich auch die Möglichkeit haben, ein neues Modul anzulegen. Hierzu sollte eine extra Seite angelegt werden, auf der der Nutzer alle relevanten Informationen zu einem Modul eintragen kann. Diese Seite sollte um die Möglichkeit ergänzt werden, dass auch ein Bild zu einem Modul mit hochgeladen werden kann, um anderen Interessierten die Möglichkeit zu bieten, direkt einen visuellen Eindruck von der Komponente bekommen zu können.

Somit hat sich das junge, aber strebsame und motivierte Team im vergangenen Wintersemester 19/20 der Aufgabe angenommen ein Webportal für PV-Kennlinien zu entwickeln, das diesen Anforderungen entspricht. Im Folgenden ist der Projektbericht für dieses Vorhaben einzusehen, welcher die Herausforderungen, Errungenschaften und zahlreiche Einzelheiten sowie wichtige Implementierungsdetails aufzeigt und den Fortschritt über den Entwicklungszeitraum dokumentiert.

## 2 Vorstellung der Gruppenmitglieder

### 2.1 Vorstellung der Gruppenmitglieder

In unserem Projektteam waren vier Mitglieder beteiligt – Hannes Rüffer, Moritz Withöft, Michelle Vorwerk und Christiane Zolkin. Jedes Teammitglied hat im Wintersemester 2017 mit dem Informatik Studium begonnen, wodurch sich eine gemeinsame Grundwissensbasis ergab.

Hannes Rüffer, der ein gutes technisches und logisches Verständnis mit sich bringt, ist in der Lage einen Überblick über alle Projektanforderungen zu bewahren und Ansätze für die Entwicklung zu finden. Durch seine Arbeitsmoral ist er ein guter Helfer und Problemlöser und seine Stärke liegt darin, das Projekt auch bei Schwierigkeiten voranzutreiben.

Moritz Withöft ist ebenfalls ein sehr motivierter Mitarbeiter und Projektleiter. Seine Stärke besteht in seiner Zuverlässigkeit, aufgetragene Aufgaben effektiv zu bewältigen. Durch seinen Nebenberuf durfte er bereits Erfahrungen im Bereich Webengineering sammeln, und konnte dies in diesem Projekt gut einsetzen.

Auch Michelle Vorwerk konnte bereits durch beruflicher Erfahrung viel zum Projekt beitragen. Durch kreieren von Weboberflächen hat sie ein Auge für Ästhetik und Pragmatik von Webseiten entwickelt, was sehr hilfreich für die Projektentwicklung war.

Christiane Zolkin ist eine zielorientierte Persönlichkeit und außerdem lernwillig, um neue Aufgaben souverän bewältigen zu können. Dadurch konnte sie zu dem Projekt einige Aspekte beitragen und Teilbereiche auf sich nehmen.

### 2.2 Aufgabenverteilung im Team

Durch die Stärken, Schwächen und unterschiedlichen, individuellen Eigenschaften aller Teammitglieder, konnten wir die Rollen innerhalb des Projektes gut einteilen.

Die Projektleiterrolle und Rolle des Dokumentationsmanagers wurden an Moritz Withöft übergeben, aufgrund seiner Zuverlässigkeit, Motivation und Sprachbegabung.

Wegen ihres Überblicks des Projektgeschehens, war Michelle Vorwerk Schnittstellen-Manager, um alle Teilbereiche der Entwicklung strukturiert zusammenzuführen.

Da das Projekt sehr Frontend-orientiert war, entwickelten Moritz, Michelle und Christiane gemeinsam an diesen Teilbereich.

Das Backend war auch sehr arbeitsintensiv, aber durch Hannes gutes Verständnis für diesen Bereich, konnte er diese Aufgabe gut übernehmen.

Alles in allem herrschte aber eine gute Gruppendynamik, und die Rollenverteilung innerhalb des Projektes war nicht statisch, sondern eher interaktiv, wodurch viele Probleme gelöst werden konnten, indem man sich einander half.

### 3 Theoretische Grundlagen

Unser Projekt ist ein Webportal zum Anzeigen und Vergleichen von Photovoltaik-Modulen und deren Daten. Um zu verstehen, wofür die Abkürzungen und Grafiken, die in unserem Portal überall wiederzufinden sind, stehen, und was diese bedeuten, soll in diesem Abschnitt erläutert werden.

#### 3.1 Photovoltaik-Zelle

Es gibt viele verschiedene Arten von Photovoltaik-Zellen, zum Beispiel

- Monokristallines Silizium,
- Polykristallines Silizium (Si),
- Amorphes Silizium (a-Si),
- Kadmium-Tellurid (CdTe) und
- Kupfer-Indium-(Gallium-)Diselenid (CIS/CIGS).

Das am weitesten verbreitete und bekannteste Material ist Silizium, da es vergleichsweise günstig, einfach zu bekommen und einfach zu verarbeiten ist. Silizium hat eine besondere Eigenschaft, welche es zu dem zentralen Bestandteil einer Photovoltaik-Zelle macht: Es ist ein Halbleiter. In Halbleitern können durch die Zufuhr von Energie, zum Beispiel in Form von energiereichem Licht oder elektromagnetischer Strahlung, in ihrem Inneren freie Ladungsträger erzeugen. Das ist allerdings nicht ohne einen weiteren kleinen Trick möglich.

Eine Silizium-Photovoltaik-Zelle hat mehrere Schichten. Die obere Schicht ist mit einem Elektronenspender (zum Beispiel Phosphoratomen) durchsetzt, während die untere Schicht mit einem Elektronenakzeptor (zum Beispiel Boratomen) durchsetzt ist. Die obere Schicht hat somit das Verlangen, Elektronen abzugeben, während die untere Schicht Elektronen aufnehmen möchte. Durch diese Diskrepanz entsteht ein ständig vorhandenes elektrisches Feld. Bis zu diesem Punkt findet der Prozess auch ohne jegliche Sonneneinstrahlung statt, aber nur durch das vorhandene elektrische Feld kann noch kein Strom gewonnen werden. Trifft nun Sonnenlicht (oder anderes energiereiches Licht) auf die Photovoltaik-Zelle, übertragen die Lichtphotonen ihre Energie an die locker gebundenen Elektronen in der Schicht des Elektronenakzeptors. Die getroffenen Elektronen lösen sich aus ihrer Bindung und begeben sich in das Leitungsband, eine Schicht zwischen den beiden Siliziumschichten. Viele der Elektronen verschwinden nach kurzer Zeit wieder, da sie sich mit immer noch vorhandenen Teilchen rekombinieren. Einige Elektronen werden durch das vorhandene elektrische Feld auf die gegenüberliegende Schicht gedrängt, während die „Löcher“, dort, wo die Elektronen herkamen, leer bleiben. Es entsteht ein Ladungsunterschied, eine Spannung. Solange weitere Lichtphotonen kontinuierlich weitere Elektronen aus ihren Bindungen lösen, entsteht ein nutzbarer Strom, der über Leiterbahnen an der oberen und unteren Siliziumschicht wieder auf die ursprüngliche Seite geführt wird.

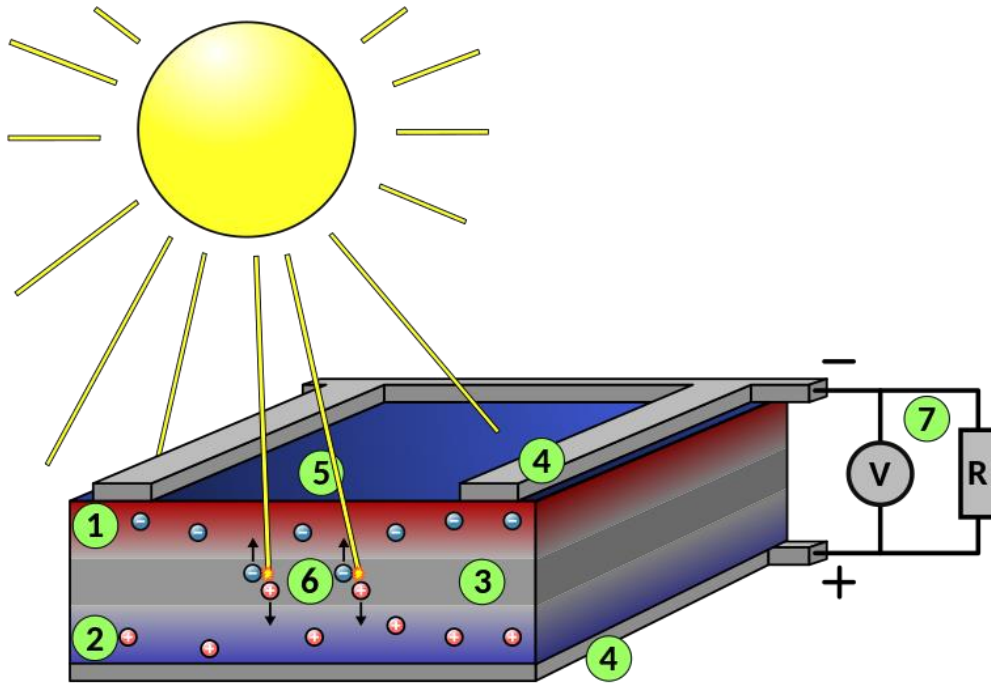


Abbildung 1: Aufbau einer Photovoltaik-Zelle<sup>5</sup>

### 3.2 MPP – Maximum Power Point

Der Punkt der maximalen Leistung (eng. „Maximum Power Point“, kurz „MPP“) ist der Punkt, an dem die Leistung einer Photovoltaikanlage unter den Standard-Testbedingungen (d.h. 25°C Modultemperatur, 1000 W/m<sup>2</sup> Bestrahlungsstärke und einer Luftmasse von 1,5 AM) maximal wird. Durch systematisches Ausprobieren von verschiedenen Widerständen auf der Leiterbahn des Verbrauchers, kann eine Folge von

Punkten gemessen werden. Diese Punkte bestehen aus einer Spannung und einer Stromstärke. Da die Leistung in Watt gleich der Spannung mal die Stromstärke ist, kann für jeden der eingesetzten Widerstände die maximal mögliche Leistung berechnet werden. Der Punkt, an dem die Leistung am höchste ist, ist der MPP – der Punkt der maximalen Leistung. Zeichnet man die Folge der Punkte in ein Diagramm und verbindet die Punkte, so erhält man den geometrischen Zusammenhang: Der Punkt der maximalen Leistung ist der Punkt, an dem die Fläche unter dem Graphen (der blaue Graph in **Abbildung 2** maximal wird. Zeichnet man das Produkt aus Spannung und Stromstärke zusätzlich, so kann man den MPP direkt ablesen, da dieser dann der Hochpunkt des neuen Graphen (pink in **Abbildung 2**) ist.

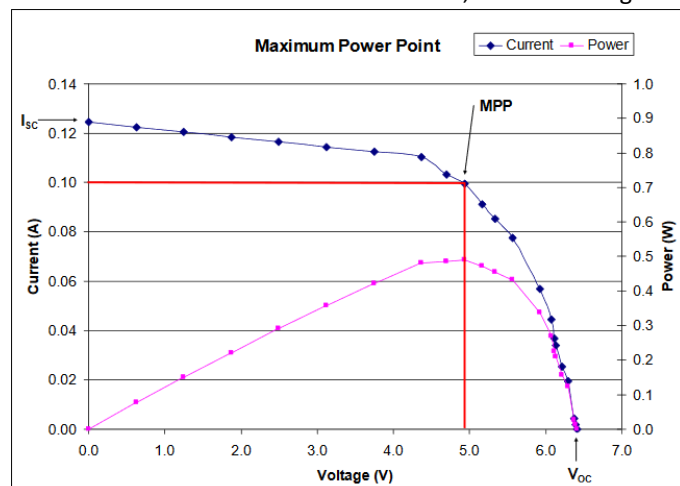


Abbildung 2: Kennlinie zum MPP

<sup>5</sup> (Dietrich, 2020)



### 3.3 $U_L$ – Leerlaufspannung

Die Leerlaufspannung  $U_L$ , im Englischen abgekürzt als  $V_{oc}$  („open circuit voltage“), wird dann erreicht, wenn keine externe Verbindung zwischen den Polen der Photovoltaik-Zelle angeschlossen ist, und die Photovoltaik-Zelle trotzdem von der Sonne bestrahlt wird. An diesem Punkt ist die höchste Spannung der Photovoltaik-Zelle erreicht, allerdings fließt dann auch kein Strom mehr. Ohne Verbindung zwischen den Polen kann kein Strom fließen, jedoch sorgt die Sonneneinstrahlung weiterhin dafür, dass Elektronen in der Siliziumschicht ihre Bindungen lösen. Da die Elektronen keinen anderen Weg mehr gehen können, um das entstandene Potenzial auszugleichen, ist die höchstmögliche Spannung erreicht.

In der **Abbildung 2** liegt die Leerlaufspannung  $V_{oc}$  bei ca. Voltage = 6,4V, am rechten, unteren Rand des Diagramms.

### 3.4 $I_K$ – Kurzschlussstrom

Der Kurzschlussstrom  $I_K$ , im Englischen abgekürzt als  $I_{sc}$  („short circuit current“), wird dann erreicht, wenn eine widerstandslose externe Verbindung zwischen den Polen der Photovoltaik-Zelle besteht. An diesem Punkt fließt der größtmögliche Strom, allerdings existiert keine Spannung mehr. Wenn kein Widerstand zwischen die Pole geschaltet ist, führt das dazu, dass jegliche Spannung, die erzeugt wird, sofortig die gegenüberliegende Schicht erreicht.

In **Abbildung 2** liegt der Kurzschlussstrom  $I_{sc}$  bei ca. Current = 1,25A, am linken, oberen Rand des Diagramms.

### 3.5 Kennlinien

Eine Kennlinie ist eine Abbildung von der Spannung, die in einer Photovoltaik-Zelle herrscht, zu der Stromstärke, die die Photovoltaik-Zelle erzeugt. Wie in 3.2 schon angesprochen, kann man durch das kontinuierliche Ändern des Widerstandes, der an der verbindenden Leiterbahn zwischen den zwei Polen liegt, eine Änderung der Spannung sowie der Stromstärke messen. Fängt man also mit kaum einem Widerstand an, erhält man einen Messpunkt nahe des Kurzschlussstroms  $I_K$ . Anschließend wiederholt man die Messung mit immer größeren Widerständen und merkt sich auch diese Messpunkte. Hat man dann einen zu hohen Widerstand erreicht, fängt die Photovoltaik-Zelle an, rapide an Stromstärke zu verlieren (ungefähr um den MPP herum). Je höher daraufhin der Widerstand wird, desto näher kommt man der Leerlaufspannung  $U_L$ , da ein zu hoher Widerstand den gleichen Effekt hat, wie als hätte man überhaupt keine Verbindung zwischen den Polen aufgebaut: Die Elektronen nehmen den Weg mit dem geringsten Widerstand, welcher dann eher durch das Innere der Photovoltaik-Zelle verläuft, als durch die Luft oder eine Verbindung mit zu hohem Widerstand.

Alle Messwerte, die während der gesamten Zeit aufgezeichnet wurden, ergeben dann die Kennlinie. Es gibt allerdings auch unter den Kennlinien verschiedene Arten: Hellkennlinien und Dunkelkennlinien.

### 3.6 Hellkennlinien

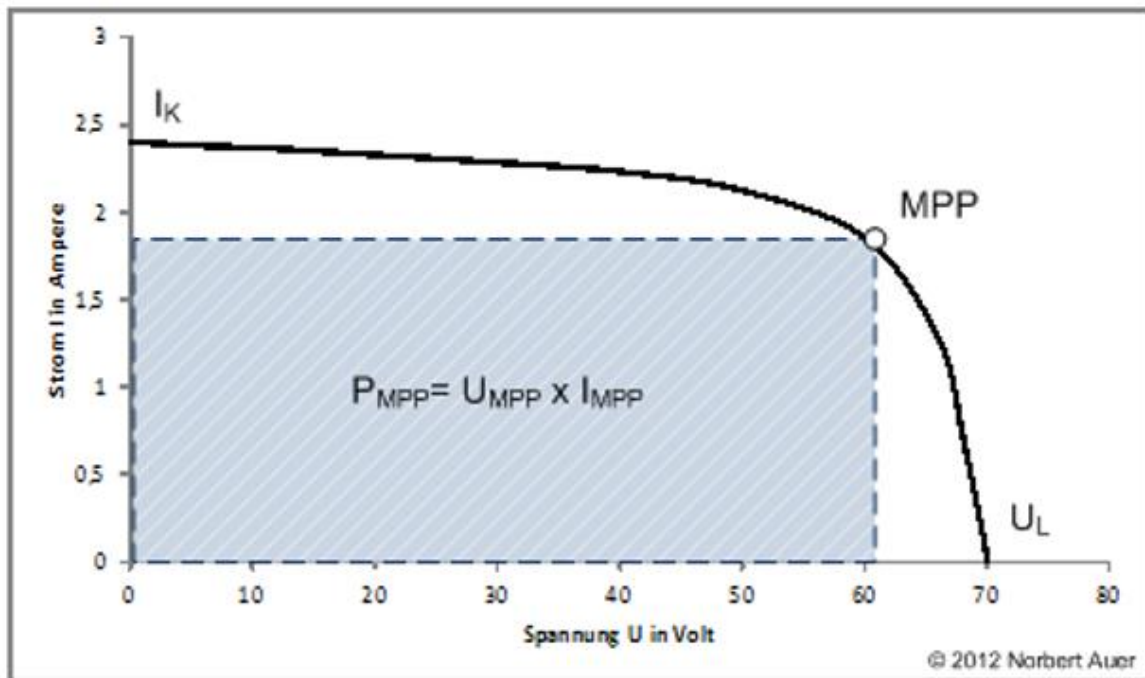


Abbildung 3: Eine Hellkennlinie<sup>6</sup>

Abgebildet in **Abbildung 3** sieht man eine exemplarische Hellkennlinie. Sie enthält alle wichtigen Punkte  $I_K$ , MPP und  $U_L$ .

Der größte Unterschied zwischen den Kennlinienarten ist, wie sie gemessen werden. Eine Hellkennlinie wird unter den Standard-Bedingungen (25°C Modultemperatur, 1000 W/m<sup>2</sup> Sonneneinstrahlung, 1,5 AM) gemessen. Diese Bedingungen zu erreichen ist nicht immer einfach, da diese sehr spezifisch und abhängig vom lokalen Wetter sind. In manchen Ländern der Erde mag es sogar auf natürlichen Weg nicht möglich sein, die Standard-Bedingungen einzuhalten. Angefertigt werden Hellkennlinien vom Hersteller selbst immer nach der Produktion von einer Photovoltaik-Zelle, am besten sollten die Messungen aber in gewissen Zeitabständen von dem Endnutzer wiederholt werden. Hellkennlinien bieten nämlich neben der Möglichkeit, die maximale Leistung leicht ablesen zu können, auch noch die Möglichkeit, gewisse Mängel an den Photovoltaik-Zellen festzustellen. So kann zum Beispiel ein hoher gemessener serieller Widerstand bedeuten, dass eine Steckverbindung fehlerhaft ist. Ein zu geringer Widerstand hingegen kann heißen, dass lokale Kurzschlüsse existieren. Solche Mängel sollten schnell behoben werden, denn sie wirken sich unvermeidbar negativ auf die Gesamtleistung einzelner Photovoltaik-Zellen, und gegebenenfalls dann auch auf ganze Stränge aus.

<sup>6</sup> (Solarstrom, 2020)

### 3.7 Dunkelkennlinien

Im Gegensatz zu Hellkennlinien werden Dunkelkennlinien<sup>7</sup> nicht unter den Standard-Bedingungen aufgenommen. Analog zu ihrem Namen werden Dunkelkennlinien nur bei einer Sonneneinstrahlung von  $0 \text{ W/m}^2$  gemessen. Sie sind sehr beliebt, da die Bedingungen für die Aufzeichnung vereinfacht sind – nicht mehr wetter-, ort- und landabhängig. Tatsächlich bieten Dunkelkennlinien auch manche Vorteile, die Hellkennlinien bieten: Durch eine Transformation der Dunkelkennlinie entlang der Y-Achse soweit, bis die Leerlaufspannung  $U_L$  bei Stormstärke  $A = 0$  liegt, erhält man wieder eine Hellkennlinie. So kann auch von einer Dunkelkennlinie der Punkt der maximalen Leistung abgelesen werden.

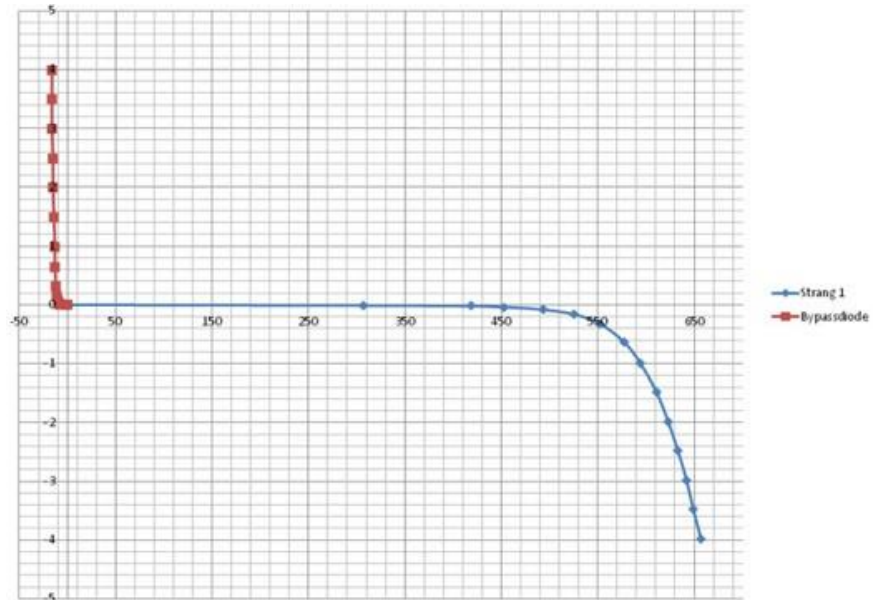


Abbildung 4: Eine Dunkelkennlinie

Auch in der Fehleranalyse sind Dunkelkennlinien beliebt. Durch das Heranziehen von Elektrolumineszenzbildern<sup>8</sup> von den Strängen, können zusammen mit der Dunkelkennlinie schnell fehlerhafte Stränge identifiziert

werden. In gewissen Fällen kann durch ein Elektrolumineszenzbild auf einen Blick gesehen werden, wo das Problem des Strangs ist. Aber auf eine Hellkennlinie kann man trotzdem nicht immer verzichten. Ist zum Beispiel das Aufnehmen der Dunkelkennlinie nicht möglich, aber die Bypassdiode kann gemessen werden, liegt eine Unterbrechung zwischen den Photovoltaik-Zellen oder zwischen der Anschlussdiode und Photovoltaik-Zellen vor. In dem Fall muss man am Tag und unter den Standard-Bedingungen nochmal eine Messung durchführen, um herauszufinden, wo genau die Unterbrechung liegt.



Abbildung 5: Ein Elektrolumineszenzbild

<sup>7</sup> (photovoltaikbüro, 2020)

<sup>8</sup> (photovoltaikbüro, 2020)

### 3.8 Nutzen des Modulportals

Das Modulportal bietet neben der großartigen Übersicht und Navigierbarkeit für erfahrene Nutzer und Photovoltaik-Enthusiasten auch eine Hilfestellung für Neu- und Noch-Nicht-Kunden. So gibt es in der Übersicht einen Bereich für Module nach ihrem Beliebtheitsgrad, wodurch Neukunden sich gezielt nach guten Modulen erkundigen können. Außerdem gibt es einen Bereich für die am meisten kommentierten Module. Diese sind sehenswert, da eine rege Diskussion höchst wahrscheinlich gute sowie schlechte Aspekte von Photovoltaik-Modulen aufdecken wird. Durch den zeitlichen Filter von Kennlinien kann auch der Performanceabfall von einzelnen Modulen betrachtet werden, wodurch Neukunden ein Modul schon aussortieren können, falls es bei einer Vielzahl von Leuten schnell an Leistung verliert. Natürlich können jegliche Benutzer auch Fragen unter Module schreiben, die von erfahrenen Benutzern beantwortet werden können.

Insgesamt bietet das Modulportal einen großen Nutzen für erfahrene Benutzer, die ihre Erfahrungen dokumentieren, austauschen und verewigen wollen, aber auch für neue Benutzer, die nur nach Ratschlägen und Hilfe suchen.

## 4 Konzeptionelle Arbeiten

Im Folgenden sind einzelne Ausschnitte der für das Webportal für PV-Kennlinien erstellte Diagramme genauer beschrieben. Auf Grund des Großen Umfangs und der Anzahl vieler verschiedener Diagramme und Diagrammtypen hat man sich hier auf wesentliche Auszüge beschränkt und diese erläutert. Alle Diagramme für das Projekt sind in voller Ausführung im Anhang vorzufinden.

### 4.1 Usecasediagramm

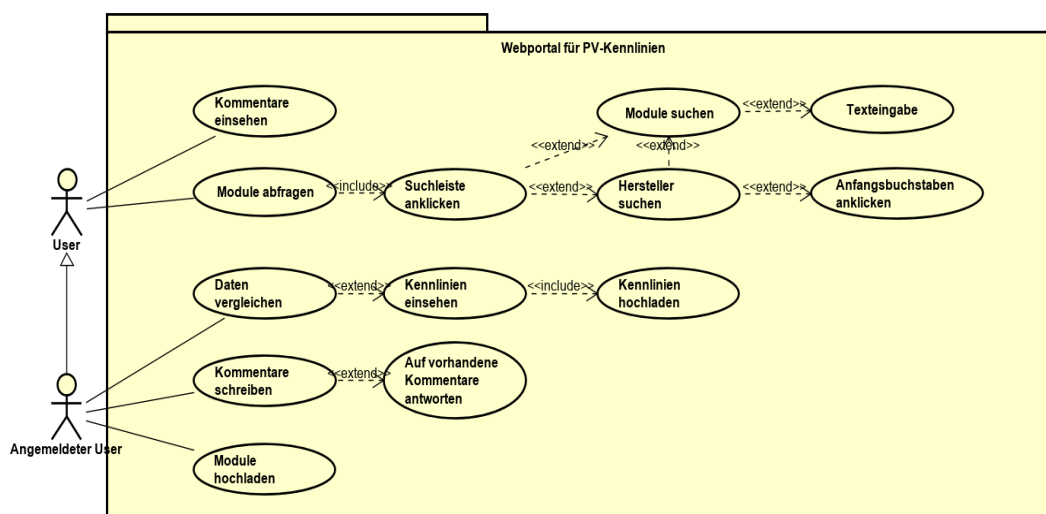


Abbildung 6: Usecasediagramm

Um alle Anforderungen übersichtlich darzustellen, wurde zunächst ein Use-Case Diagramm angelegt. Damit konnte man abgleichen, was der Auftraggeber erwartet, und somit was die Entwickler zu implementieren haben. Missverständnisse wurden somit schnell identifiziert und die Anforderungen konnten gut geändert werden.

Wichtig sind dabei eine Modulübersicht, sowie eine Modulabfrage über ein Suchfeld. Es ist möglich nach speziellen Modultypen zu suchen, allerdings aber auch nach Herstellern und deren Produkte. Außerdem möchte man Daten mehrerer Module vergleichen können, indem man Module auswählen kann und auf eine Detailseite weitergeleitet wird. Dort werden technische Daten des Moduls und Kennlinien übersichtlich angezeigt. Außerdem gibt es eine Kommentarsektion, in der Diskussionen und Fragen von Kunden geklärt werden können, indem man Kommentare lesen, aber auch schreiben kann.

Des Weiteren sollten auch neue Modultypen von angemeldeten Nutzern hochgeladen werden können.

## 4.2 Sequenzdiagramm

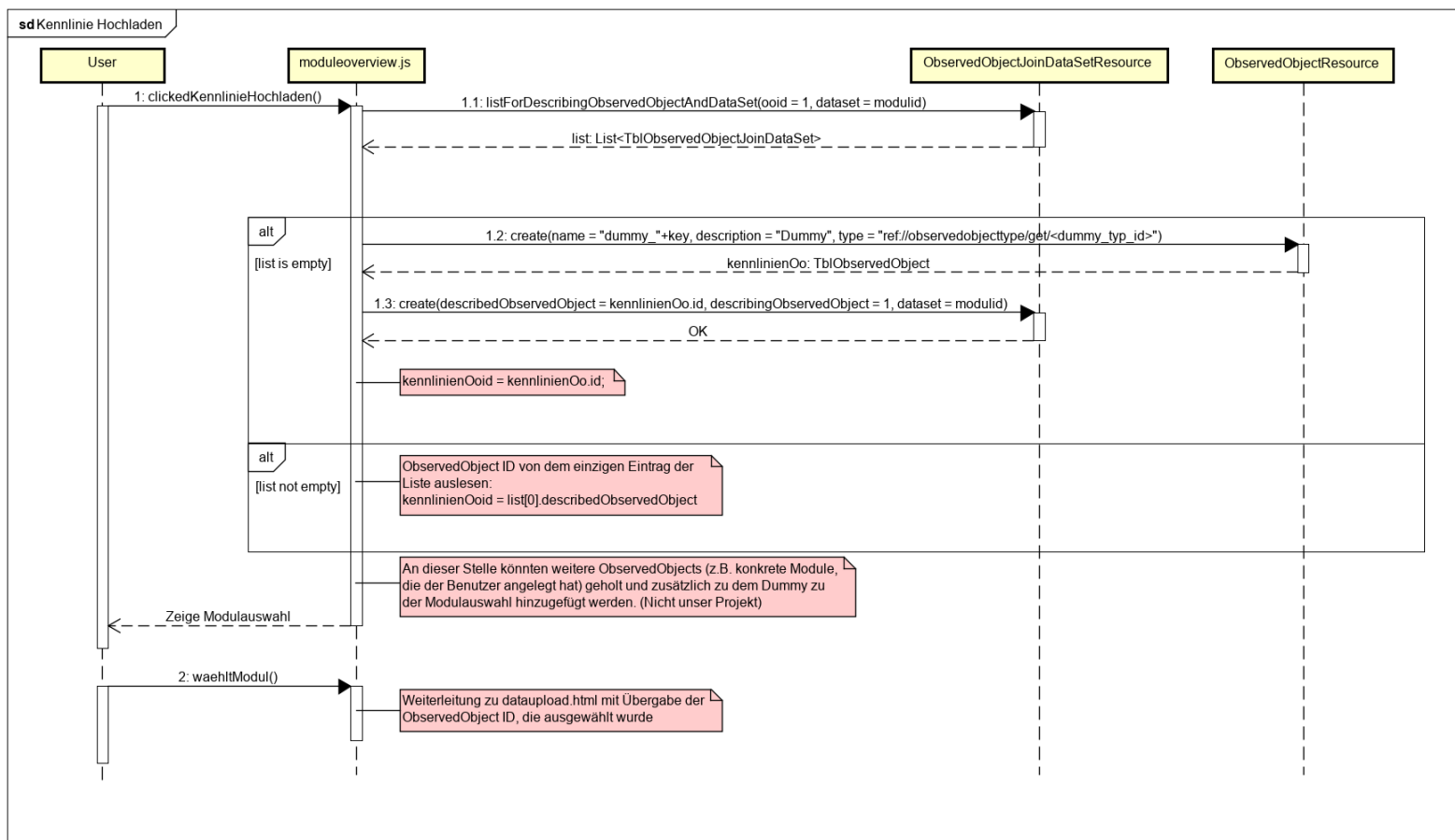


Abbildung 7: Sequenzdiagramm zum Kennlinienupload

Um Prozess des Kennlinien-Uploads wurde zu einem besseren Überblick ein Sequenzdiagramm erstellt. Ausgelöst wird dieser Prozess, indem der User „Neue Kennlinie hinzufügen“ anklickt. Über die Schnittstelle „listDescribingObservedObjectAndDataSet“ wird geprüft, ob ein Dummy ObservedObject für den

Kennlinien-Upload bereits existiert. Falls ja, kann die Kennlinien-Dummy-ID ausgelesen werden und zur Upload-Seite weitergeleitet werden.

Wenn die zurückgegebene Liste allerdings keine Einträge vorweist, muss ein Kennlinien-Dummy-Objekt zunächst einmal angelegt werden. Dies geschieht über die Schnittstelle „create“ der `ObservedObjectResource`. Danach wird über die „create“-Schnittstelle der `ObservedObjectJoinDataSetResource` eine Verbindung zwischen dem neuem `ObservedObject` und Modul-Objekt geschaffen. Wenn keine Probleme bei dem Kreieren aufgetreten sind, wird nun zur Upload-Seite übergeleitet.

In der Zukunft können noch weitere Dummy-ObservedObjects angelegt werden, um beispielsweise zu konkreten Modulen Kennlinien hochladen zu können. Diese Funktionalität ist allerdings noch nicht vorhanden.

### 4.3 Klassendiagramm

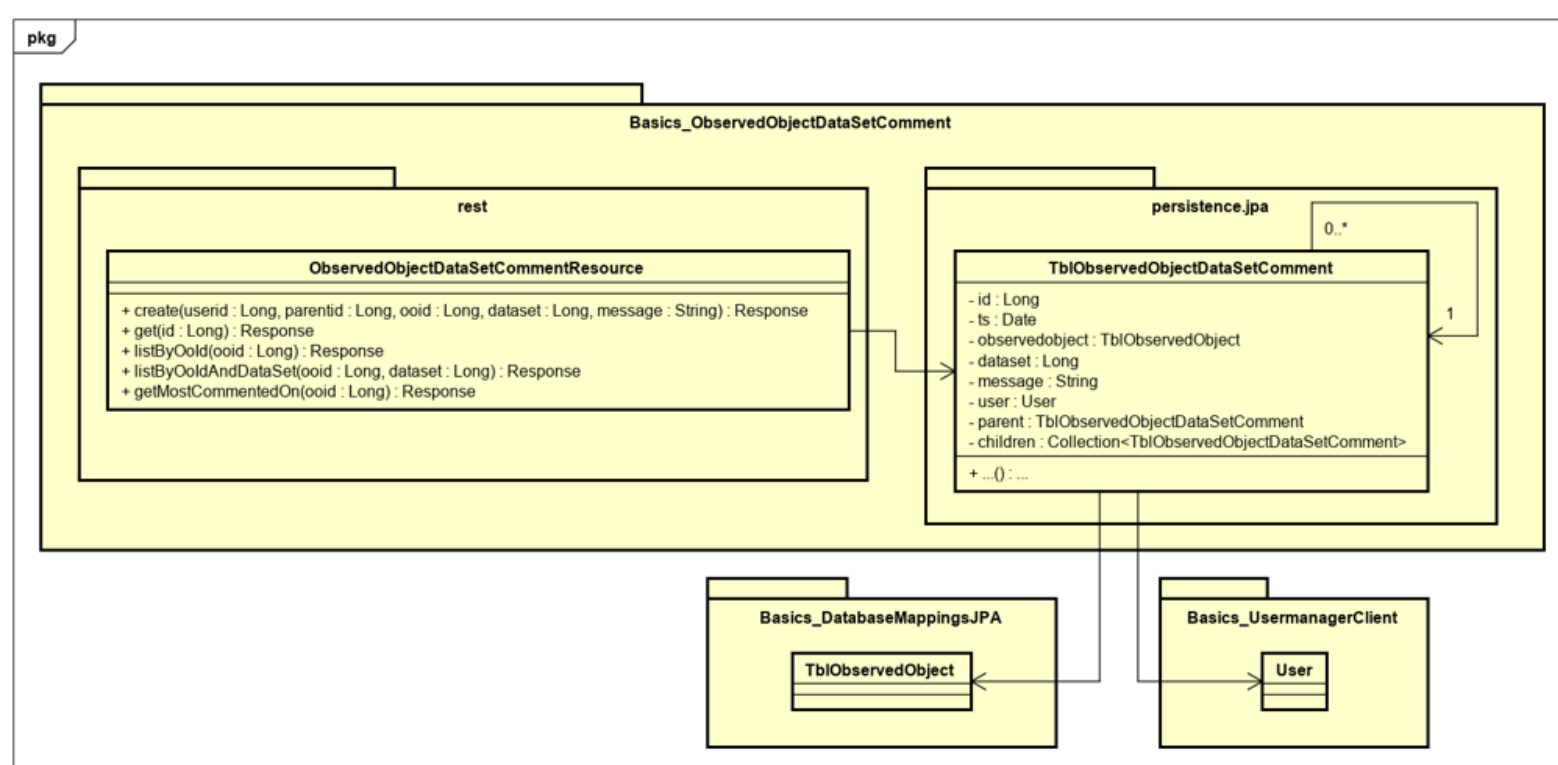


Abbildung 8: Klassendiagramm der Kommentare

Die ursprüngliche Planung für die Kommentarfunktion im Backend sah so aus, dass ein neues Projekt erstellt wird, in dem es jeweils ein Paket für die REST-Schnittstelle und die JPA-Klasse gibt. Die Idee war, an den Trend der Microservices anzuknüpfen, und so ein eigenständiges Projekt zu haben, welches so unabhängig wie möglich von den anderen Projekten ist. So sieht man in der obigen Abbildung, dass nur zwei Referenzen benötigt werden, um das Projekt zu realisieren: Eine Referenz auf die Speicherklass für die `ObservedObjects` und eine auf die Benutzer Klasse.

So kann die Speicherklass der Kommentare alle Informationen erreichen, die benötigt werden: „ts“ als Zeitstempel, zu dem der Kommentar angelegt wird, „observedobject“ als Referenz auf das entsprechende `ObservedObject`, „dataset“ als optionaler Zusatz, zu welchem Datensatz der Kommentar gehört, „message“ als Nachricht des Kommentars, „user“ als Referenz auf das User Objekt, von welchem aus der Kommentar

angelegt wurde, „parent“ als Referenz auf einen weiteren Kommentar, um eine hierarchische Abbildung der Kommentare zu schaffen, „children“ als Gegenstück zu der „parent“-Referenz.

Die REST-Schnittstelle für die Kommentare beinhaltet dann die Standardfunktionalität „create“ und „get“ (welche in der Implementierung noch um „list“ und „delete“ erweitert wurden), sowie diverse Schnittstellen, um spezifische Gruppen von Kommentaren zu laden.

In der Umsetzung von dem Diagramm hat es leider ein paar unlösbare Probleme gegeben: Die REST-Schnittstelle benötigt eine „PersistenceUnit“ um von außen erreichbar zu sein, jedoch kann keine neue „PersistenceUnit“ für die Kommentare angelegt werden, da man dann keinerlei Zugriff mehr auf jegliche vorhandenen Objekte hätte, das heißt auch keinen Zugriff auf ObservedObjects, die aber benötigt werden. Dazu kam noch, dass man auch das Hauptprojekt „SmartMonitoringBackend“ in dem neuen Projekt nicht verlinken konnte, da sonst eine „zyklische Referenz“ entstünde, die allerdings von Netbeans verboten ist. Man kann aber das Hauptprojekt auch nicht **nicht** verlinken, weil das Projekt für die Kommentare sonst nicht zur Runtime des Servers verfügbar ist.

Gelöst wurde das Problem auf eine weniger schöne, aber funktionierende Art, indem die REST-Schnittstelle in der Hauptprojekt bewegt wurde. Die JPA-Klasse konnte in dem neuen Projekt verbleiben. So gab es keine zyklische Referenz mehr und der Server hat zur Runtime alle Ressourcen und Klassen gefunden.

## 5 Code-Implementierungsdetails

Im Folgenden werden zu verschiedenen Code-Auszügen Implementierungsdetails vorgestellt. Dabei wird nicht auf alle Einzelheiten des Webportals für PV-Kennlinien eingegangen, sondern auf wesentliche Aspekte. Diese Aspekte haben sich im Laufe der Entwicklung als erwähnenswert herausgestellt, weil sie entweder Besonderheiten aufweisen oder essenziell für die korrekten Funktionsweisen des Portals sind.

### 5.1 Änderung von Queryparametern

Soll eine Suche durchgeführt werden, muss dazu eine Anfrage vom Frontend an das Backend geschickt werden, die den Suchstring beinhaltet. Um jetzt eine Range-Query durchzuführen (der String kann an beliebiger Stelle im Namen vorkommen und muss nicht am Anfang stehen) wird ein sogenannter Wildcard-Operator benötigt, der in Datenbanken durch das „%“-Zeichen realisiert wird. Dabei tritt allerdings das Problem auf, dass bei GET-Anfragen Strings mit diesem Zeichen durch HTML kodiert werden, sodass beispielsweise aus „%AB“ „%AB%27“ wird. Um dieses Problem zu umgehen und noch immer die richtige Suchanfrage an das Backend senden zu können, musste der Wildcard-Operator neu definiert werden und wurde so durch ein „\$“ ersetzt. Im Backend muss dieser Schritt des Austauschs wieder rückgängig gemacht werden. Eine Suchanfrage aus dem Frontend an das Backend mit Wildcard-Operator ist in folgendem Beispiel zu sehen.

```
if (SWAC_modulesearch.searchString.length >= SWAC_modulesearch.minimumSearchInput) {  
    let fetchURLmodules = `/SmartMonitoringBackend/data/getSets?ooid=1&searchcolumns=modul&searchvalues=${SWAC_modulesearch.searchString}$`;  
    SWAC_modulesearch.fetchData(fetchURLmodules, SWAC_modulesearch.combinedCallbackModules);  
} else {  
    UIKit.notification('Mindestens 3 Zeichen eingeben.', {status:'warning'});  
}
```

Abbildung 9: Codeauszug Änderung von Queryparametern

## 5.2 Suche im Backend

Die Suchfunktionalität ist im Backend in zwei Schritten implementiert: Parsen der Anfrage und Korrektes einfügen in die Datenbankabfrage.

```
Map<String, Object> likes = null;
if (searchcolumns != null && searchvalues != null) {
    List<String> searchColsList = new ArrayList<>(Arrays.asList(searchcolumns.split(",")));
    List<String> searchValuesList = new ArrayList<>(Arrays.asList(searchvalues.split(",")));
    //Error Handling { ... }

    boolean sameSearchValues = false;
    if(searchValuesList.size() == 1){
        sameSearchValues = true;
    }

    likes = new HashMap<>();
    for(int i = 0; i < searchColsList.size(); i++){
        if(sameSearchValues){ //Always use the first entered search value
            likes.put(searchColsList.get(i), searchValuesList.get(0));
        } else {
            likes.put(searchColsList.get(i), searchValuesList.get(i));
        }
    }
}
```

Abbildung 10: Codeausschnitt: Parsen der Suchanfrage

Abbildung 6 ist ein verkürzter Ausschnitt der Methode „getSets(...)“ der Klasse „DataResource“. Beim Parsen der Anfrage ist zu beachten, dass die Suche auf zwei Parameter aufgeteilt ist. Der erste Parameter „searchcolumns“ ist eine kommaseparierte Folge von Spaltennamen, in denen man Suchen möchte. Der zweite Parameter „searchvalues“ ist entweder ein einzelner Wert (d.h. er enthält keine Kommata), nach welchem dann in allen Spalten gesucht wird, oder auch eine kommaseparierte Folge von Suchbegriffen. In dem letzteren Fall muss die Anzahl der Spaltennamen gleich der Anzahl der Suchbegriffe sein, ansonsten gibt die REST-Schnittstelle einen Fehler zurück. Anschließend werden die Suchspalten und -begriffe in einer „HashMap<String, String>“ einander zueingordnet.

```
if (likes != null && !likes.isEmpty()) {
    if (wheres != null && !wheres.isEmpty()) {
        sqlbuilder.append(" AND ");
    }

    int count = 0;
    //TODO replace with implode
    sqlbuilder.append("(" );
    for (String curLike : likes.keySet()) {
        sqlbuilder.append("LOWER(\"").append(curLike).append("\") LIKE LOWER(:");
        sqlbuilder.append(curLike);
        sqlbuilder.append("\");");
        if (count < likes.size() - 1) {
            sqlbuilder.append(" OR ");
        } else {
            sqlbuilder.append(" )");
        }
        count++;
    }
}
```

Abbildung 11: Codeausschnitt: Einfügen der Suchbegriffe in die PreparedQuery



Ist das Parsen der Anfrage erfolgreich, wird die Anfrage an die „DynamicTable“ des angefragten ObservedObjects weitergeleitet. **Abbildung 7** zeigt den Ausschnitt der Methode „getPreparedQuery(...)“ der Klasse „DynamicTable“, in dem die Suchparameter in den SQL-String geschrieben werden. Es wird für jeden Eintrag der Such-„HashMap“ lediglich der Spaltenname in dieser Form eingesetzt: „LOWER(<spaltenname>) LIKE LOWER(:<spaltenname>)“. Der erste <spaltenname> wird aufgelöst zu dem tatsächlichen Spaltennamen, während der zweite <spaltenname> zusammen mit dem Doppelpunkt ein Platzhalter für die Bindung zu dem Wert zu einem späteren Zeitpunkt ist. Beide Spaltennamenersetzungen sind von einem „LOWER()“ umschlossen, das heißt, dass die Suche sowohl bei der Spalte als auch bei dem Wert die Groß- und Kleinschreibung nicht berücksichtigt. Dazwischen steht ein „LIKE“, was in SQL-Sprachen für einen Ähnlichkeitsvergleich mit möglichen „Wildcards“ steht. Nach jedem, außer dem letzten, Eintrag in der Such-„HashMap“ wird ein „OR“ angehängen, was dazu führt, dass ein Datensatz gefunden wird, wenn auch nur eine Spalte aus der angefragten Suche den entsprechenden Suchbegriff beinhaltet. Es wurde überlegt, ob man die Suche dynamischer gestalten sollte, das heißt die „LOWER(...)“ abschaltbar, bzw. das „OR“ durch ein „AND“ tauschbar macht, was in der Theorie auch nicht zu schwer zu realisieren gewesen wäre, allerdings haben wir uns dagegen entschieden, weil jede weitere Option als Parameter mit an die Funktion übergeben werden muss. Da die Funktion bereits 11 Parameter hat, würden zwei weitere Parameter, die bei den meisten Aufrufen der Methode sowieso „null“ sind, der Übersichtlichkeit, und damit auch der Wartbarkeit und Erweiterbarkeit des Programms schaden.

Die „Wildcards“ bei dem Ähnlichkeitsvergleich mit dem „LIKE“ Schlüsselwort in SQL haben unvorhergesehene Probleme ausgelöst. Es gibt die zwei Arten von „Wildcards“: Ein „.“ (Punkt) steht für ein einzelnes beliebiges Zeichen, ein „%“ (Prozent) für eine beliebig lange Folge von Zeichen, auch keine. Somit könnte auf die Suche „D%“ in der Spalte „land“ die Antworten „Deutschland“, „Dänemark“ oder auch einfach nur „D“ zurückkommen. („D“ ist kein Land, aber es würde von dem Suchbegriff gefunden werden.) Problematisch wird es dadurch, dass die Suche über die GET-Parameter der Anfrage gehandhabt wird und GET-Parameter von allen modernen Browsern von jeglichen nicht-ASCII-Zeichen bereinigt werden. Anstelle der nicht-ASCII-Zeichen ersetzt ein Browser eine Kodierung beginnend mit einem Prozentzeichen, gefolgt von einer Zahl, die für das entsprechende Zeichen steht. Das Zeichen „Ü“ würde in dieser Kodierung die Zeichenfolge „%C3%9C“ haben. Wenn der Benutzer aber nach einem Modul sucht, wird sein Suchbegriff von Prozentzeichen umschlossen. Der Suchbegriff „C3%9C“ würde also umgeformt werden zu „%C3%9C%“. Auf der Serverseite kann diese Kodierung wieder rückgängig gemacht werden, allerdings kann der Server nicht unterscheiden, ob der Browser oder der Benutzer ein Zeichen kodiert hat. Der Server dekodiert alle Sonderzeichen, die er findet. Das führt dazu, dass gewisse Zeichen auf der Serverseite dekodiert werden, die dazu führen, dass der Server abstürzen kann. Um das zu umgehen, wird sowohl im Frontend als auch im Backend das „Wildcard“ Zeichen durch das Paragraphenzeichen ersetzt, und nach der Übertragung wieder zu dem Prozentzeichen zurückersetzt.

```
// Setting likes
if (likes != null) {
    for (Entry<String, Object> curLike : likes.entrySet()) {
        query.setParameter(curLike.getKey(), curLike.getValue().toString().replaceAll("$", "%"));
    }
}
```

Abbildung 12: Codeausschnitt der Wildcard-Ersetzung im Backend

### 5.3 Handling der Suchergebnisse

Nachdem eine Suche durchgeführt wurde und muss die Antwort im Frontend verarbeitet werden. Dazu wurde die Funktion `addTableResult` geschrieben, die die Suchergebnisse der Tabelle hinzufügt. Da das Suchergebnis allerdings entweder aus Modultypen oder aus Herstellern bestehen kann, wurde die Funktion anwendungsunabhängig geschrieben, sodass dieser neben den Suchergebnissen auch die Tabelle als Selektor übergeben werden muss, in der die Ergebnisse eingefügt werden sowie, ob es sich um ein Modul oder einen Hersteller handelt.

Diese Unterscheidung wird durch ein *Enum* realisiert und ist wichtig, da der Nutzer bei einem Klick auf ein Modul auf die Moduldetailseite weitergeleitet werden soll. Bei einem Klick auf einen Hersteller, soll jedoch eine weitere Suche nach Modulen, vom Herstellernamen ausgehend, durchgeführt werden.

```
SWAC_modulesearch.addTableResult = function (dataset, resultDiv, type, id) {
    dataset.forEach(element => {
        let tableData = document.createElement("td");
        tableData.innerText = element;
        tableRow.appendChild(tableData);
    });
    switch (type) {
        case SWAC_modulesearch.resultType.MODULE:
            $(tableRow).click(function() {
                window.location = `moduledetail.html?id=${id}`;
            });
            break;
        case SWAC_modulesearch.resultType.MANUFACTURER:
    }
```

Abbildung 13: Codeauszug Handling der Suchergebnisse

### 5.4 Kennlinien anzeigen

```
/**
 * @module ModuleDetail
 */
var kennlinienOoid = 0;
var dummy_type_id = 0;
var active_kennlinie = [];
var active_kennlinie_index = 0;
var module_name = "";
var kennlinien_type_index = 0;
var kennlinien_types = [];
var kennlinien_types_data = [];
```

Abbildung 14: Code Abschnitt - Kennlinien anzeigen 1

Um Kennlinien in einem Diagramm anzeigen zu können, werden Datenstrukturen gebraucht, um die von der Datenbank geholten Daten zu speichern und somit verarbeiten zu können. Hierzu werden Variablen angelegt.

Zur Speicherung der Kennlinien zugehörig zu einem Modultypen, wurde ein Dummy Type ObservedObject angelegt. Dessen ID (dummy\_type\_id) wird gefetcht, um das referenzierte ObservedObject – die zugehörigen Kennlinien also – herauszufinden (kennlinienOOid).

Die restlichen Variablen sind Datenstrukturen für die Speicherung der von der Datenbank geladenen Kennlinien. Dabei handelt es sich bei „kennlinien\_types\_data“ um ein dreidimensionales Array, welches die Kennlinien-Typen, die Kennlinien-Datensätze und die Punkte einer einzelnen Kennlinie enthält.

„active\_kennlinie“ stellt dabei die aktuell angezeigte Kennlinie im Diagramm dar. Sie wird ermittelt über „kennlinien\_type\_index“ und „active\_kennlinie\_index“.

```
/**
 * Gets the DescribedObservedObject (the curves that are referenced by the
 * selected module) and saves the ID of it.
 */
function loadKennlinien() {
    console.log("loadKennlinien()");

    $.ajax({
        method: "GET",
        url: '/SmartMonitoringBackend/observedobjectjoindataset/listForDescribingObservedObjectAndDataSet?oid=1&dataset=' + dataset
    }).always(function (msg) {
        var c_index = msg.list[0].describedObservedObject.match(/\d+/g);
        kennlinienOOid = c_index[0];
        readKennlinien();
    });
}
```

Abbildung 15: Code Abschnitt - Kennlinien anzeigen 2

Mit dem Laden der Seite wird die Funktion „loadKennlinien()“ initiiert. Mit der Modul ID werden dann die DescribedObservedObjects geladen und die Kennlinien ID gespeichert. Nun können die Kennlinien ausgelesen werden.

```
/**
 * Gets the curve data and saves the different curve types.
 */
function readKennlinien() {
    var kennlinien_id = 0;
    $.ajax({
        method: "GET",
        url: '/SmartMonitoringBackend/observedobject/listChilds?parent_id=' + kennlinienOOid + '&recursive=false'
    }).always(function (msg) {
        console.log("readKennlinien()");
        kennlinien_types = msg.list;

        for(let i = 0; i < kennlinien_types.length; i++) {
            kennlinien_id = msg.list[i].id;
            $.ajax({
                method: "GET",
                url: '/SmartMonitoringBackend/data/getSets?oid=' + kennlinien_id
            }).always(function (data) {
                saveData(data.list);
            });
        }
    });
}
```

Abbildung 16: Code Abschnitt - Kennlinien anzeigen 3

Die Kennlinien ObservedObject ID ermöglicht das Auslesen aller Kennlinien eines Modultypen. Mit der Schnittstelle „observedobject/listChilds“ werden alle in der Datenbank gespeicherten Kennlinien Typen zurückgegeben – Hellkennlinien, Dunkelkennlinien und Laborkennlinien. Jeder Eintrag in der zurückgelieferten Liste stellt einen Kennlinientypen dar, und bei jedem Eintrag handelt es sich um ein Array mit mehreren Kennlinien Datensätzen. Diese Daten werden nur mit der Funktion „saveData()“ gespeichert und für das Diagramm verfügbar gemacht.

```
/**
 * Converts the curve data and saves it, so it can be displayed in the chart.
 * @param {type} list
 */
function saveData(list) {
    var kennlinie = [];
    var kennlinien = [];
    for(let j = 0; j < list.length; j++) {
        for (let i = 1; i <= 250; i++) {
            let select_i = "I" + i;
            let select_u = "U" + i;
            if (list[j][select_u] !== null && list[j][select_i] !== null) {
                kennlinie.push({
                    id: i,
                    u: list[j][select_u],
                    i: list[j][select_i],
                    p: list[j][select_u] * list[j][select_i]
                });
            }
        }
        kennlinien.push(kennlinie);
        kennlinie = [];
    }
    kennlinien_types_data.push(kennlinien);
    active_kennlinie = kennlinien_types_data[kennlinien_type_index][active_kennlinie_index];
}
```

Abbildung 17: Code Abschnitt - Kennlinien anzeigen 4

In „saveData()“ werden die zurückgelieferten Kennliniendaten mit einer doppelten for-Schleife durchiteriert. Die äußere for-Schleife ist für die Iteration durch die Kennlinien-Typen verantwortlich, die innere für das Durchlaufen aller Datensätze des aktuellen Kennlinien-Typs.

Dementsprechend wird ein dreidimensionales Array zur Speicherung aller Daten herangezogen – um Kennlinien-Typen, Kennlinien-Datensatz und Kennlinien-Punkte abspeichern zu können und die Daten an das Diagramm zu überliefern. Angezeigt wird dann die „active\_kennlinie“, welche über den Kennlinien-Type-Index und den Aktiven-Kennlinie-Index (beide anfangend bei 0) ermittelt wird.

### 5.4.1 Kennlinien hochladen

```
/**
 * Initiates the curve upload to the module.
 */
function uploadKennlinie() {
    console.log("uploadKennlinie()");
    remoteHandler.fetchGet("observedobjectjoindataset/listForDescribingObservedObjectAndDataSet", {ooid: 1, dataset: dataset}, false)
        .then(listForDescribingObservedObject_callback).catch(function (error) {
            UIKit.notification({
                message: 'Keine Verbindung vorhanden' + error,
                timeout: SWAC_config.notifyDuration,
                pos: 'top-center',
                status: 'error'
            });
        });
};
}
```

Abbildung 18: Kennlinien hochladen 1

Sobald „Neue Kennlinie hinzufügen“ auf der Nutzeroberfläche angeklickt wird, wird die Funktion „uploadKennlinie()“ ausgeführt.

Der Upload besteht aus mehreren Schritten: Zunächst wird über die Schnittstelle „observedobjectjoindataset/listForDescribingObservedObjectAndDataSet“ die Liste aller vom Modultypen referenzierten Kennlinien zurückgegeben, die über die Callback-Funktion „listForDescribingObservedObject\_callback()“ verarbeitet werden kann.

```
/**
 * Callback function from listing the DescribedObservedObjects. If the list does
 * not contain any entries, a new ObservedObject is created to save the curve data in.
 * If there are entries, the IDs of the DescribedObservedObjects will be shown
 * in the SelectBox of the PopUp Window to upload a new curve.
 * @param {type} data is the list of DescribedObservedObjects
 */
function listForDescribingObservedObject_callback(data) {
    console.log("listForDescribingObservedObject_callback()");

    if(data.list.length === 0) {
        //Wenn keine Liste vorhanden, wird ObservedObjectType Dummy erstellt
        remoteHandler.fetchPost("observedobject/create", {name: module_name + "_Kennlinien_" + dataset,
            description: "Observed Object zum Speichern von Kennlinien, die zu einem Modultyp gehören.",
            type: "ref://observedobjecttype/get/"+dummy_type_id, dataCapture: true}, false)
            .then(createDummy_callback).catch(function (error) {
                });
    }
    else {
        addKOOIdToSelectBox();
    }
}
```

Abbildung 19: Kennlinien hochladen 2

In der Callback-Funktion wird nun geprüft, ob in dieser Liste Einträge vorhanden sind – also, ob bereits ein Kennlinien-ObservedObject vorhanden ist, bei dem neue Datensätze hinzugefügt werden können. Falls nicht, wird dieses ObservedObject erstmals angelegt. Ansonsten wird der User mit der zurückgegebenen Kennlinien-OOID zur Upload-Seite weitergeleitet.

```

/**
 * Callback function from creating the dummy. The ID of the newly created
 * ObservedObject is saved.
 * @param {type} data
 */
function createDummy_callback(data) {
    console.log("createDummy_callback()");
    kennlinienOoid = data.id;
    //Verbindung von Modul zu Kennlinien erstellen
    remoteHandler.fetchPost("observedobjectjoindataset/create",
        {describedObservedObject: data.id, describingObservedObject: 1, dataset: dataset}, false)
        .then(createDescribedOO_callback).catch(function (error) {
        });
}

/**
 * Callback function from creating the DescribedObservedObject. The current module
 * now references the curve dummy ObservedObject.
 * @param {type} data
 */
function createDescribedOO_callback(data) {
    console.log("createDescribedOO_callback()");
    addKOOIdToSelectBox();
}

```

Abbildung 20: Kennlinien hochladen 3

Wenn das Kennlinien-ObservedObject erfolgreich angelegt werden konnte, wird die „createDummy\_callback()“-Funktion ausgeführt, welche eine Referenz zwischen dem aktuellen Modultypen und dem neu kreierte Kennlinien-ObservedObject schafft. Falls auch dies erfolgt, kann der User zur Upload-Seite des Kennlinien-Dummys weitergeleitet werden. Nach dem Hochladen können nun die hochgeladenen Kennlinien beim nächsten Aufruf der Moduldetail-Seite angezeigt werden.

## 5.5 Modulübersicht

Die Module in der Modulübersicht werden mithilfe der SWAC Komponente Cardpresenter angezeigt. Die meist besuchten Module werden mit der „observedobjectdatasetviewrequest/getMostViewed“ REST Schnittstelle geholt, die neusten Module mit der „data/getSets“ REST Schnittstelle, wobei hier alle Module geholt und sortiert werden, und die meist kommentierten mit der „observedobjectdatasetcomment/getMostCommentedOn“ REST Schnittstelle. Alle Daten werden hierbei auf 8 Module beschränkt.

```

<div id="module-overview-most-viewed" class="uk-container">
    swa="swac_cardpresenter FROM observedobjectdatasetviewrequest/getMostViewed WHERE ooid=1 AND limit=8">
    <h2 class="uk-heading-small uk-heading-divider">Meist besuchte Module</h2>
    <div ...24 lines />
</div>

<div id="module-overview-newest" class="uk-container">
    swa="swac_cardpresenter FROM data/getSets WHERE ooid=1 AND limit=8 AND orderby=id AND order=DESC">
    <h2 class="uk-heading-small uk-heading-divider">Neuste Module</h2>
    <div ...23 lines />
</div>

<div id="module-overview-most-commented" class="uk-container">
    swa="swac_cardpresenter FROM observedobjectdatasetcomment/getMostCommentedOn WHERE ooid=1 AND limit=8">
    <h2 class="uk-heading-small uk-heading-divider">Am meisten kommentiert</h2>
    <div ...23 lines />
</div>

```

Abbildung 21: HTML Code Modulübersicht

## 5.6 Senden eines Kommentars

```
SWAC_commentsection.saveComment = function (form) {
  let requestor = SWAC_view.findRequestor(form);

  let dataCapsle = {};
  dataCapsle.data = [];
  dataCapsle.data[0] = {};

  dataCapsle.metadata = {};
  dataCapsle.metadata.fromSource = requestor.fromName;

  ooid = requestor.fromWheres['ooid'];
  message = form.elements['comment'].value;
  dataset = requestor.fromWheres['dataset'];
  user = JSON.parse(localStorage.getItem('swac_currentUser'))['id'];
  parent = this.getParentComment(form);

  if( parent != false && parent != null ) {
    dataCapsle.data[0]['parentid'] = parent;
  }

  dataCapsle.data[0]['ooid'] = ooid;
  dataCapsle.data[0]['message'] = message;
  dataCapsle.data[0]['dataset'] = dataset;
  dataCapsle.data[0]['userid'] = user;

  let savePromise = SWAC_model.save(dataCapsle);
  savePromise.then(function (result) {
    location.reload();
  });
};
```

Abbildung 22: Senden eines Kommentars

Das Senden eines Kommentares passiert durch die Save Methode aus dem SWAC Model. Vorher werden alle wichtigen Daten in einem Objekt „dataCapsle“ zusammengefasst. Die richtige Schnittstelle bekommt man durch den Requestor, sodass die Komponente anwendungsfallunabhängig ist. Nach dem Absenden des Kommentares wird die Seite neu geladen.

## 5.7 Speichern eines Kommentars im Backend

Der Kommentar wird an die REST Schnittstelle „observedobjectdatasetcomment/create“ gesendet. Die Daten werden als JSON an die create Methode übergeben. Die Daten aus dem JSON String werden in ein Objekt geschrieben und überprüft. Danach wird ein neuer Kommentar mit den Daten erstellt und in die Datenbank geschrieben.

```

@POST
@Path("create")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Response create( String json ) {
    User user = null;
    TblObservedObjectDataSetComment parent = null;
    TblObservedObject oo = null;
    Long dataset = null;
    String message = null;

    ResponseObjectBuilder rob = new ResponseObjectBuilder();

```

Abbildung 24: Speichern eines Kommentars 1

```

    TblObservedObjectDataSetComment newComment = new TblObservedObjectDataSetComment();
    newComment.setTimeStamp(Calendar.getInstance().getTime());
    newComment.setParent(parent);
    newComment.setObservedobject(oo);
    newComment.setDataset(dataset);
    newComment.setMessage(message);
    newComment.setUser(user);

    try{
        utx.begin();
        this.em.persist(newComment);
        utx.commit();

        rob.setStatus(Response.Status.CREATED);
        rob.add(newComment);
    }catch(IllegalStateException | SecurityException
           | HeuristicMixedException | HeuristicRollbackException
           | NotSupportedException | RollbackException
           | SystemException ex){
        rob.setStatus(Response.Status.INTERNAL_SERVER_ERROR);
        rob.addErrorMessage("Could not create new comment due to an exception: "+ex.getLocalizedMessage());
    }

    return rob.toResponse();
}

```

Abbildung 23: Speichern eines Kommentars 2

## 5.8 Backend Media

Eine Teilaufgabe des Backends war es, die bereits vorhandene Media-Schnittstelle umzuschreiben. Die vorherige Media-Schnittstelle hat eine „1:n“-Beziehung implementiert, diese sollte nun zu einer „n:m“-Beziehung umgeschrieben werden. Die Kritik an der alten Version war, dass wenn man das gleiche Bild zu verschiedenen ObservedObjects hochladen möchte, dieses auch tatsächlich zweimal in der Datenbank ablegen muss. Das führt dazu, dass unnötig Ressourcen von dem Server verbraucht werden. Die neue „n:m“-Beziehung löst dieses Problem, indem beliebig viele Verbindungen zwischen einzelnen Medien und ObservedObjects angelegt werden können, ohne doppelten Platz zu verbrauchen.



Dazu kommt, dass das Modulportal eine neue Anforderung an Medien hatte: Es sollte möglich sein, zu einem Bild nicht nur ein ObservedObject, sondern auch eine Datensatz ID hinterlegen zu können. Ein Modultyp in unserer Datenbank ist lediglich ein Datensatz eines ObservedObjects, jedoch sollen verschiedene Modultypen auch verschiedene Bilder haben können. Umgesetzt wurde das dadurch, dass sowohl in der „TblMedia“-Klasse, als auch in der „TblObservedObject“-Klasse die Referenzen auf die jeweils andere Ressource entfernt wurde und eine neue Verbindungsentität angelegt wurde. „TblMedia“ und „TblObservedObject“ enthalten dann jeweils eine Liste von Objekten der Verbindungsentität.

```
@OneToMany(mappedBy = "media", fetch = FetchType.LAZY)
private Collection<TblMediaJoinObservedObjectDataSet> mediaobservedobjects;
```

Abbildung 25: Neues Feld in der "TblMedia"-Klasse

Die Verbindungsentität namens „TblMediaJoinObservedObjectDataSet“ hat dann jeweils einen Eintrag für ein Media- und ein ObservedObject-Objekt. Zusätzlich hat die Verbindungsentität ein optionales Feld für die ID des Datensatzes, an den das Bild gebunden sein soll.

```
@Basic(optional = false)
@ManyToOne
@JoinColumn
private TblMedia media;

@Basic(optional = false)
@ManyToOne
@JoinColumn
private TblObservedObject observedobject;

@Column(name = "dataset", nullable = true)
private Long dataset;
```

Abbildung 26: Felder in der Verbindungsentität zwischen Media und ObservedObject

## 5.9 Backend View-Requests

Die größten Teile des Backends für die View-Requests ist simpel: Eine Speicherklass für die Daten, eine Schnittstelle für die Standardoperationen und der Rest wird von JAX-RS und Hibernate geregelt. Zwei Problemstellen gab es allerdings, die erste davon bei der Abfrage, welcher Datensatz eines ObservedObjects am meisten angeschaut wurde. Hibernates Methoden, um Queries zu erzeugen können nur typisiert sein, wenn genau **ein** Typ zurückgegeben wird, sei es ein primitiver Typ oder eine Klasse. In diesem Fall müssen allerdings zwei Werte zurückgegeben werden: Die Datensatz ID, nach der **gruppiert** wird, und die Anzahl der Besuche, nach der **sortiert** wird.

```

//Returns a list in the form of: List { Object[] {Long dataset1, Long view_count1}, Object[] {Long dataset2, Long view_count2} }
List<Object[]> mostViewed = this.em.createNamedQuery("TblObservedObjectDataSetViewRequest.getMostViewedByOoId")
    .setParameter("oo", requestedObject)
    .setFirstResult(startset.intValue())
    .setMaxResults(limit)
    .getResultList();

//Used to track the order of the most viewed datasets
List<Long[]> datasetOrders = new ArrayList<>();

//Parse the result to the datasetOrders
for(Object entry : mostViewed){
    Object[] entryArr = (Object[]) entry;

    Long dataset = (Long)entryArr[0];
    Long viewCount = (Long)entryArr[1];

    datasetOrders.add(new Long[] {dataset, viewCount});
}

```

Abbildung 27: Abfrage der meist besuchten Module aus der Datenbank

Um das zu realisieren, muss man eine nicht-generische Query erstellen und ausführen. Dadurch bekommt man eine nicht-generische Liste zurück, über die man manuell iterieren und die einzelnen Werte von „Object“ zu, in diesem Fall, „Long“ parsen muss. Ein unschöner Weg, aber leider das einzige, was Hibernate in diesem Fall anbietet.

Die zweite Problematik bestand darin, dass mehrere Datensätze, die nicht unbedingt hintereinander liegen, zurückgegeben werden müssen. Würden diese hintereinander liegen, könnte man durch ein gut gewähltes „OFFSET“ und „LIMIT“ seine Datensätze bekommen, das findet hier aber keine Anwendung. Man müsste also in die SQL-Abfrage in die „WHERE“ Bedingung die IDs der Datensätze mit aufnehmen, zum Beispiel so: „... WHERE id = ,1‘ OR id = ,40‘ OR id = ,19001‘ OR ...“. Allerdings ist die „WHERE“ Bedingung in der Schnittstelle der „DynamicTable“ bereits implementiert und die „WHERE“ Bedingungen werden mit „AND“ getrennt, nicht mit „OR“. Dadurch kann man immer noch maximal einem Datensatz gleichzeitig fragen, was in unserer Umsetzung auch genau so passiert ist.

```

Map<Long, Map<String, Object>> allRows = new HashMap<>();
Map<String, Object> wheres = new HashMap<>();

for(Long[] datasetOrder : datasetOrders){
    wheres.put("id", datasetOrder[0]);
    allRows.put(datasetOrder[0], dt.getSingleDataset(measurementnamesCol, wheres, null));
}

ResponseListBuilder rlb = new ResponseListBuilder();
for(Long[] datasetOrder : datasetOrders){
    Map<String, Object> row = allRows.get(datasetOrder[0]);
    row.put("view_count", datasetOrder[1]);
    rlb.add(row);
}

rob.add("list", rlb);

```

Abbildung 28: Einzelabfragen aller gefundenen Datensätze

## 5.10 Explizite Verarbeitung von Bildern

In dem vorliegenden Code-Ausschnitt ist eine Funktion zu sehen, die Daten in Form eines Bildes erhält und einen Selektor bekommt, in dem das Bild eingefügt werden soll, damit es für den Nutzer auf der Seite sichtbar ist. Diese Daten hat man von dem Backend bekommen, als man ein Bild für ein Modul per GET-Request angefordert hat. Jedoch hat diese Antwort einen kleinen Fehler mit einer hohen Auswirkung, wodurch es notwendig wurde, dass das erhaltene JSON explizit zu einem Objekt verarbeitet werden muss. Denn in diesem JSON befindet sich ein versteckter Zeilenumbruch, der über die *Replace*-Funktion von JavaScript vorher entfernt werden muss. Aus diesem Grund ist es zusätzlich auch notwendig, dass man JQuery explizit mitteilt, dass Text (*dataType: „text“*) empfangen wird, da es sonst zu einem Fehler kommt. Anschließend ist ersichtlich, dass das JSON problemlos verarbeitet werden kann und das darin enthaltene Bild an richtiger Stelle auf der Seite eingefügt werden kann. Dieser Fehler wurde für „preview“ nun behoben, besteht aber immer noch, wenn das Bild in voller Auflösung angefordert wird.

```
SWAC_cardpresenter.fetchData = function (fetchURL, callback, image) {  
    //Grabs the data from the backend  
    $.ajax({  
        type: "GET",  
        url: fetchURL,  
        dataType: "text",  
        success: function (data, textStatus, request) {  
            callback(data, image);  
        },  
        error: function (e, f) {  
            console.log(`Error: ${f}`);  
            console.log(e);  
            console.log(fetchURL);  
        }  
    });  
};
```

Abbildung 29: Codeauszug Explizite Verarbeitung von Bildern

## 6 Ausblick und Fazit

### 6.1 Ausblick

Als weiteres Feature sollte es möglich sein, Kennlinien zu einem bestimmten Modul, anstatt nur zu einem Modultyp hochzuladen. Des Weiteren muss die Cardpresenter Komponente mit der Presenter Komponente vereint werden, da beide Komponenten fast identisch sind.

### 6.2 Fazit

Die Planung zu Beginn war zum Teil etwas schwierig, da die Einarbeitung in das vorhandene Projekt viel Zeit in Anspruch genommen hat und es Missverständnisse bei den Anforderungen gab. Doch die Erstellung des Grob- und Feinkonzeptes war trotzdem sehr hilfreich.

Durch die gute Kommunikation und Gruppendynamik blieben wir immer auf dem aktuellen Stand und konnten dadurch auch unsere Meilensteine gut einhalten.

Der Puffer am Ende unseres Gantt-Diagramms war gut geplant, da wir wegen der Feiertage zwei unserer eigenen Meilensteine nach hinten verschieben mussten.

## 7 Installationshinweise und Benutzerhandbuch

Im Folgenden soll mit Hilfe der Installationshinweise und des Benutzerhandbuchs die Nutzung und Bedienung der PV-Modul Webportals beschrieben werden. Hierbei wird nur die Nutzung aus Nutzersicht beschrieben, wobei Hintergrundaspekte sowie Erläuterungen zur Umsetzung in der Implementierung außer Acht gelassen werden.

Dabei wird bei den Installationshinweisen darauf eingegangen, wie das Webportal lokal zur Ausführung gebracht werden kann, wobei im Benutzerhandbuch die Nutzung eines bereits lauffähigen und funktionierenden Webportals im Fokus steht.

### 7.1 Installationshinweise

Die Installation und Benutzung kann nur einwandfrei auf einem Server mit einer frisch aufgesetzten Datenbank funktionieren, da die IDs der benötigten ObservedObjects hardkodiert sind. Die Möglichkeit, diese IDs dynamisch, auf eine Art, die nicht jede Menge Programmieraufwand hat, abzufragen, besitzt SWAC noch nicht.

Die Installation des Modulportals folgt nach der normalen Installation des SmartMonitoring-Frontends und -Backends. Es muss lediglich folgende drei Punkte gemacht werden:

- Das zusätzliche Projekt „Basics\_ObservedObjectDataSetComment“ aus GitLab klonen und in den Ordner „projects“ im SmartMonitoring Ordern ablegen
- In der Datenbank manuell die kryptische Einschränkung in der Tabelle „tbl\_measurement\_type“ löschen
- Zu der Seite „/SmartMonitoring/sites/modulesetup.html“ navigieren, auf den Knopf „Alle Ausführen“ klicken und ca. 20 Sekunden warten, bis alle Schritte der Installation automatisch abgeschlossen werden

Eine Installationsanleitung ist auch nochmal unter diesem Link zu finden:

[http://git01-ifm-min.ad.fh-bielefeld.de/Forschung/scl/2015\\_03\\_SCL\\_SmartMonitoring\\_Frontend/-/wikis/Installationsanleitung%20PVModulPortal](http://git01-ifm-min.ad.fh-bielefeld.de/Forschung/scl/2015_03_SCL_SmartMonitoring_Frontend/-/wikis/Installationsanleitung%20PVModulPortal)

## 7.2 Benutzerhandbuch

### 7.2.1 Modulübersicht

Auf die Modulübersichtsseite gelangt man in der Navigationsleiste über den Punkt *Menü -> Modulübersicht*.

Die Modulübersichtsseite ist die Startseite des Webportals für PV-Kennlinien. Diese Seite ist in drei Abschnitte unterteilt:

- Übersicht der **am meisten besuchten** Module
- Übersicht der **neuesten** Module
- Übersicht der **am meisten kommentierten** Module

Hierbei gelangen Sie durch Klicken auf das Bild oder auf die Überschrift des Moduls auf die zugehörige Moduldetailseite (siehe Abbildung 30: Modulübersicht Beispiel). Weiterhin erhalten Sie hier grundlegende Informationen über ein Modul, um es ggf. identifizieren zu können. In allen drei Abschnitten gibt es jeweils maximal zehn Module, die angezeigt werden. Sollte es weniger geben, beispielsweise nur zwei Module unter *Am meisten kommentiert*, so gibt es nur zwei Module, die bisher überhaupt einen Kommentar erhalten haben.



PMPP:	WP	900
UMPP:	V	67
IMPP:	A	34
UOC:	V	23
ISC:	A	42


Abbildung 30: Modulübersicht Beispiel

### 7.2.2 Moduldetailseite

Ist man nun auf der Modulseite angelangt, so wird man feststellen, dass diese in verschiedene Bereiche aufgeteilt ist. Im oberen Bereich gibt es die verschiedenen Informationen, auf die im Folgenden eingegangen wird. Für die verschiedenen Reiter, die sich weiter unten auf der Seite befinden, sind die Abschnitte 7.2.3, 7.2.4, 7.2.6, 7.2.7 und 7.2.7 durchzulesen.

Neben dem Bild sowie den Informationen, die der Nutzer bereits auf der Übersichtsseite erhalten hat, erhalten Sie hier weitere detaillierte Informationen zum ausgewählten PV-Modul (siehe Abbildung 31: Moduldetail Beispiel). Einige Beispiele hierfür sind die Maße des Moduls, genauere Angaben zum Hersteller sowie Herstellungsland, aber auch Informationen über die genauen Leistungseigenschaften des Moduls, die der Nutzer als Vergleichswerte zu Nutzen weiß.

Beispieltyp



Modul-Hersteller:	Beispielmodul
Modul-Typ:	Beispieltyp
Land:	Beispieland
Höhe [mm]:	30
Breite [mm]:	30
Tiefe [mm]:	30

Version:	1.0
Glasstärke Front [mm]:	3.2
Füllfaktor:	1.6
Zellhersteller:	Beispielhersteller
Nennleistung [W]:	900
Zelltyp:	Beispielzelltyp
Anzahl Zellen:	8
Zellgröße Länge [mm]:	9
Zellgröße Breite [mm]:	9
Anzahl Zellstrings:	1
Zellen pro Zellstring:	5

Anzahl Bypassdioden:	9
Serienwiderstand:	6
Parallelwiderstand:	8
Spannung im MPP [V]:	67
Leerlaufspannung [V]:	23
Strom im MPP [A]:	34
Kurzschlussstrom [A]:	42
TK Leerlaufspannung [mv/K]:	42
TK Kurzschlussstrom [%/K]:	45
TK Leistung [%/K]:	42

Abbildung 31: Moduldetail Beispiel

### 7.2.3 Kennlinien vergleichen

Damit Sie Kennlinien miteinander vergleichen können, wählen Sie auf der Moduldetailseite den Reiter *Kennlinien*. Sollten hier keine Kennlinien angezeigt werden, wurde für dieses Modul noch keine Kennlinien hochgeladen. Wie Sie Kennlinien hochladen können erfahren Sie in Abschnitt 7.2.4.

Über die Pfeile in der Navigation haben Sie die Möglichkeit zwischen Kennlinien des gleichen Typs zu wechseln, um sich verschiedene anzeigen zu lassen. Über die Reiter *Hellkennlinien*, *Dunkelkennlinien* und *Laborkennlinien* können sie zwischen Kennlinien verschiedener Typen wechseln. Zusätzlich haben Sie die Möglichkeit verschiedene Filter auf die Kennlinien anzuwenden, die sich rechts neben der Tabelle finden lassen.

### 7.2.4 Kennlinien hochladen

Möchten Sie nun auch Kennlinien hochladen, müssen Sie hierfür angemeldet sein. Unter dem Reiter *Kennlinien* auf der Moduldetailseite finden Sie die Funktion *Neue Kennlinie hochladen*. Klicken Sie auf diesen Punkt, gelangen Sie nach einer Bestätigungsabfrage auf eine neue Seite, auf der Kennlinien hochgeladen werden können.

Auf der neuen Seite *Datenupload* finden Sie nun unter *ObservedObject* bereits eine Vorauswahl, die so beibehalten werden muss, damit der Upload einer Kennlinie dem entsprechenden Modul zugeordnet werden kann. Über Drag & Drop kann man nun seine Kennlinien-Daten in das Feld ziehen oder man nutzt die Option „wählen Sie hier eine aus“. Nach einer kurzen Wartezeit, in der die hochgeladenen Daten verarbeitet werden, sollten die Kennlinien-Daten nun auch auf der Moduldetailseite angezeigt werden (7.2.3).

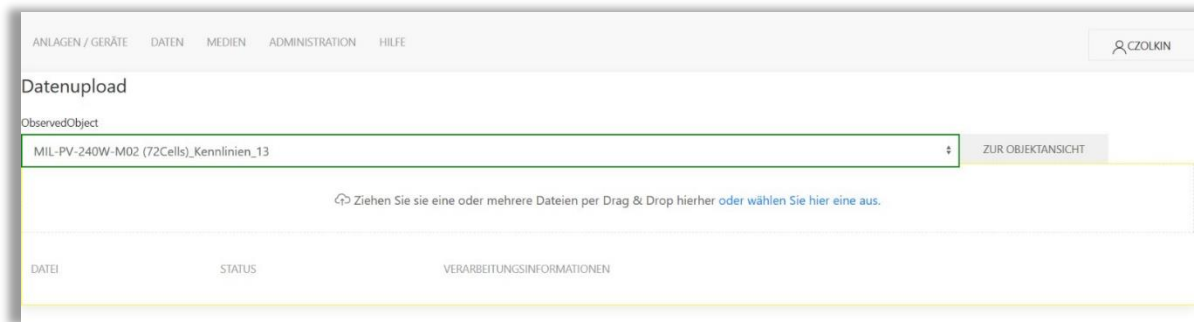


Abbildung 32: Kennlinienupload Beispiel

### 7.2.5 Fehleranalyse

Aktuell stehen die Funktionen der Fehleranalyse noch nicht zur Verfügung. Im kommenden sollen hier Analysen angezeigt werden, die Fehler an einem bestimmten PV-Modul aufzeigen und diese auswerten.

### 7.2.6 Moduldetails downloaden

Unter dem Reiter *Downloads* auf der Moduldetailseite lassen sich Informationen zum entsprechenden Modul downloaden. Hier wird dem Nutzer sowohl der Download-Typ als auch der Dateiname mitgeteilt. Auf diese Weise wissen Sie direkt um welche Information zum Herunterladen es sich handelt. Aktuell stehen hier nur die Bilder der PV-Module zum Download bereit. In Zukunft sollen Ihnen auch die Kennlinien zur Verfügung gestellt werden.

### 7.2.7 Module kommentieren

Möchten Sie Kommentare zu einem Modul einsehen, hinterlassen oder mit anderen Nutzern in eine Diskussion eintreten, so können Sie dies unter dem Reiter *Diskussionen* vollziehen. Zum Hinterlassen von Kommentaren muss der Nutzer angemeldet sein. Durch das Klicken auf *Neuer Kommentar* erscheint Ihnen ein Feld, in dem Sie einen Kommentar anlegen und anschließend abschicken können (siehe Abbildung 33: Kommentarsektion Beispiel). Weiterhin lassen sich unter bestehenden Kommentaren zwei Buttons finden. Der „Sprechblasen“-Button klappt zu einem Kommentar zugehörige Unterkommentare auf. Der „Pfeil“-Button bieten dem Nutzer die Möglichkeit auf einen bestehenden Kommentar zu antworten.

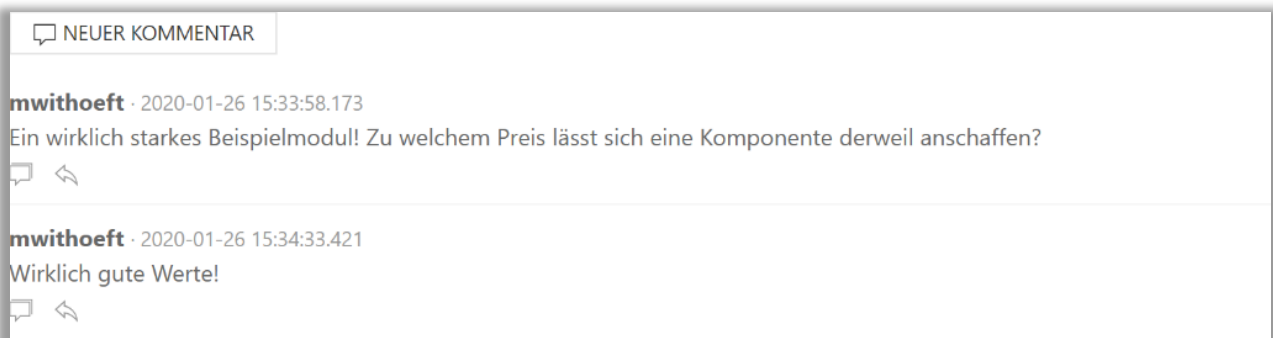


Abbildung 33: Kommentarsektion Beispiel

## 7.2.8 Suche nach Modulen

Die Suche ist auf jeder Seite des Webportals rechts oben aufzufinden. Dabei bietet die Suche zwei verschiedene Vorgangsweisen zur Findung eines Moduls:

- Volltextsuche über Modulnamen und Herstellernamen
- Suche nach Hersteller über dessen Anfangsbuchstaben

### Volltextsuche über Modulnamen und Herstellernamen:

Um eine Volltextsuche über Modulnamen und Herstellernamen durchzuführen gibt der Nutzer seinen Suchparameter in dem Eingabefeld der Suche ein. Die Eingabe kann durch einen einfachen Klick in das Suchfeld gestartet werden. Durch das Drücken der *Entertaste* oder durch einen Klick auf die *Lupe* wird die Suche begonnen. Es öffnet sich automatisch eine Tabelle mit den Suchergebnissen. Dabei werden zuerst die Module angezeigt, die den Suchparameter im Namen des Modultypen aufweisen. Danach werden die Module angezeigt, die den Suchparameter im Namen des Herstellers aufweisen. Durch einen Klick auf die Module gelangen Sie auf die Moduldetailseite.

### Suche nach Hersteller über dessen Anfangsbuchstaben:

Um die Suche nach Hersteller über dessen Anfangsbuchstaben zu starten, muss der Nutzer zunächst die *Erweiterte Suche* öffnen, indem dieser den nach unten zeigenden Pfeil neben dem Suchfeld anklickt. Danach öffnet sich eine Liste mit den Buchstaben des Alphabets. Hier können Sie Ihren gewünschten Anfangsbuchstaben anklicken, worauf darunter alle Hersteller angezeigt werden, die mit dem angeklickten Buchstaben im Namen beginnen. Durch einen Klick auf die Hersteller, öffnet sich rechts daneben eine weitere Tabelle, die alle Module des angeklickten Herstellers aufweist. Durch einen Klick auf eines der Module gelangen Sie nun zur gewählten Moduldetailseite (siehe Abbildung 34: Modulsuche Beispiel).

Suche nach Hersteller mit Anfangsbuchstaben

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Suchergebnisse Hersteller

HERSTELLER
E & H Co. Ltd.
E.ON Energie Deutschland GmbH
Earth byba
Easy BIPV APS
EC Solar Wuxi Sailing Solar

Suchergebnisse Module

MODUL-TYP	VERSION	NENNLEISTUNG [W]
Earth 290 Wp	April 2018	290
Earth 300 Wp	April 2018	300
Earth 275 Wp	April 2018	275

Abbildung 34: Modulsuche Beispiel

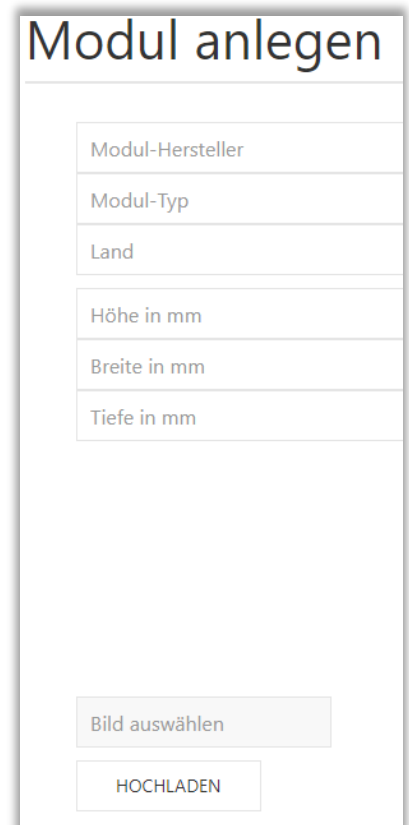


### 7.2.9 Anlegen neuer Module

Auf die Seite zum Anlegen neuer Module gelangt man in der Navigationsleiste über den Punkt *Menü -> Modul anlegen*.

Auf dieser Seite können Sie ein komplett neues Modul anlegen (siehe Abbildung 35: Ausschnitt zum Modul anlegen). Die einzelnen Felder auf der Seite beschreiben die Informationen, die der Nutzer in dem jeweiligen Feld angeben kann. Dabei sind bisher nur die Felder *Modul-Hersteller*, *Modul-Typ* und *Version* verpflichtend.

Weiterhin es Ihnen möglich, wenn vorhanden, ein Bild zum neuen PV-Modul hochzuladen, welches auf der Übersichtsseite und auf der Detailseite angezeigt würde. An dieser Stelle empfehlen wir, von dieser Möglichkeit gebrauch zu machen, da ein Bild anderen Nutzern immer eine optische Möglichkeit bietet, um sich mit dem Modul vertraut zu machen.



The screenshot shows a web form titled "Modul anlegen". It contains several input fields stacked vertically: "Modul-Hersteller", "Modul-Typ", "Land", "Höhe in mm", "Breite in mm", and "Tiefe in mm". Below these fields, there is a button labeled "Bild auswählen" and another button labeled "HOCHLADEN".

Abbildung 35: Ausschnitt zum Modul anlegen

## 8 Literaturverzeichnis

- Behrens, G. (2019). *Einführungsveranstaltung*. Minden: FH Bielefeld Solar Computing Lab.
- BMU, A. I. (2019). Klimaschutzpolitische Grundsätze und Ziele. In A. I. BMU, *Klimaschutzplan 2050* (S. 42). Berlin: Bundesministerium für Umwelt, Naturschutz und nukleare Sicherheit.
- Diermann, R. (18. Dezember 2019). *Erneuerbare Energien deckten 2019 fast 43 Prozent des deutschen Stromverbrauchs*. Abgerufen am 26. Januar 2020 von PV Magazine: <https://www.pv-magazine.de/2019/12/18/erneuerbare-energien-deckten-2019-fast-43-prozent-des-deutschen-stromverbrauchs/>
- Dietrich, S. (26. Januar 2020). *Photovoltaik*. Abgerufen am 2020. Januar 2020 von Wikipedia: <https://de.wikipedia.org/wiki/Photovoltaik>
- Lab, S. C. (31. März 2014). *Solar Computing Lab (SCL)*. Abgerufen am 26. Januar 2020 von FH Bielefeld University of Applied Sciences: <https://www.fh-bielefeld.de/minden/solar-computing-lab>
- photovoltaikbüro*. (26. Januar 2020). Von Was bringt eine Kennlinienmessung bei der Fehlersuche an PV-Anlagen?: <https://photovoltaikbuero.de/pv-know-how-blog/was-bringt-eine-kennlinienmessung-bei-der-fehlersuche-an-pv-anlagen/> abgerufen
- photovoltaikbüro. (26. Januar 2020). *Dunkelkennlinien von Photovoltaikanlagen messen*. Von photovoltaikbüro. abgerufen
- Solarstrom, P. (2020). *Der Maximum Power Point und das MPP-Tracking*. Abgerufen am 26. Januar 2020 von PhotovoltaikSolarstrom: <https://photovoltaiksolarstrom.com/photovoltaiklexikon/maximum-power-point-mpp-tracking/>

## 9 Anlagen

# PV-Modulportal - Gantt-Diagramm

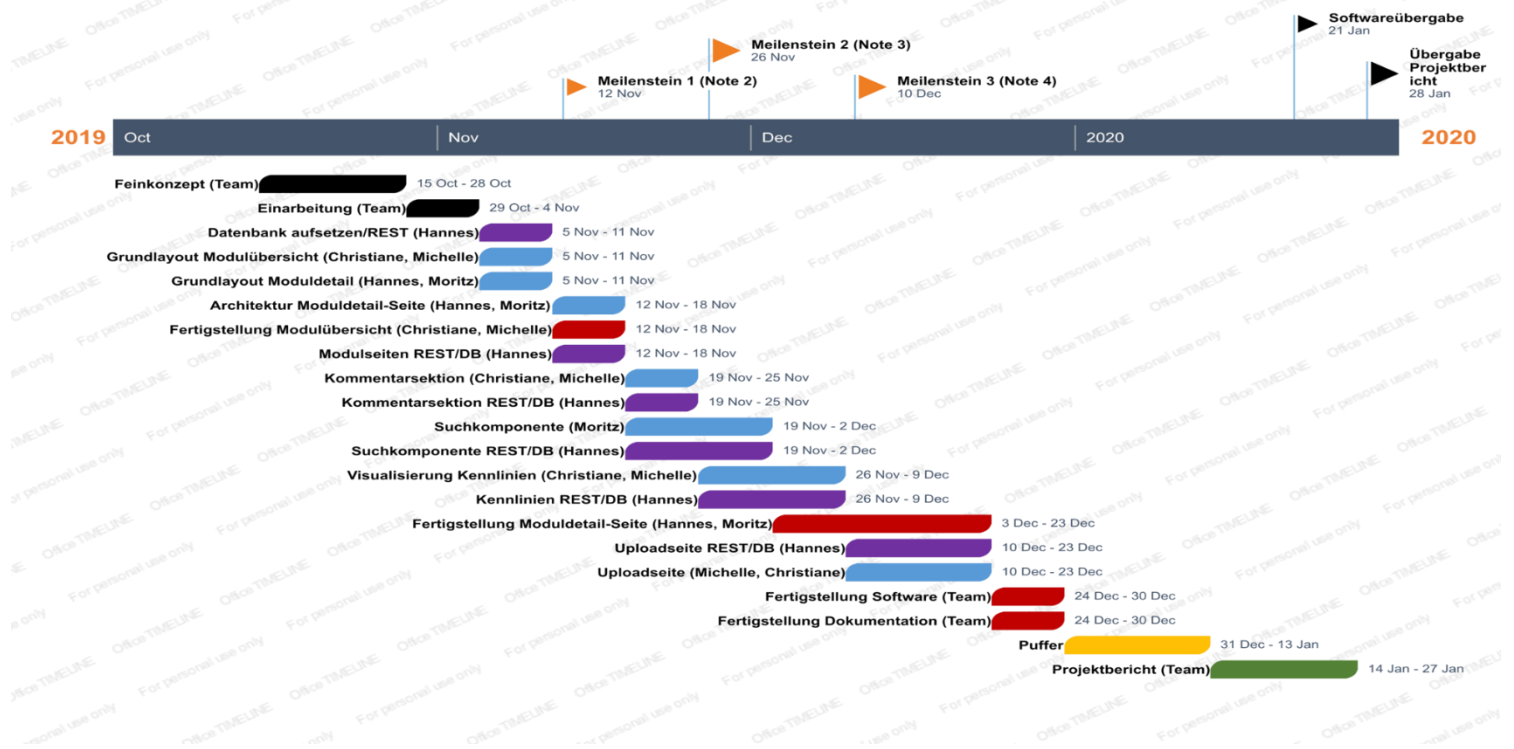


Abbildung 36: Gantt-Diagramm

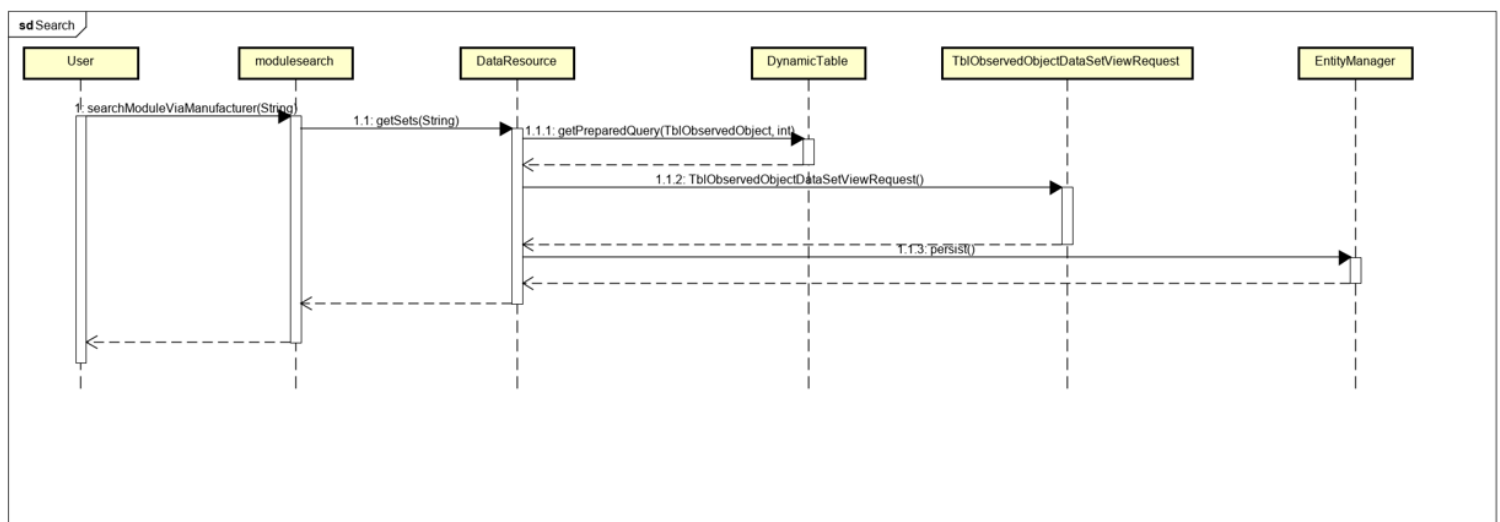


Abbildung 37: Sequenzdiagramm Suche

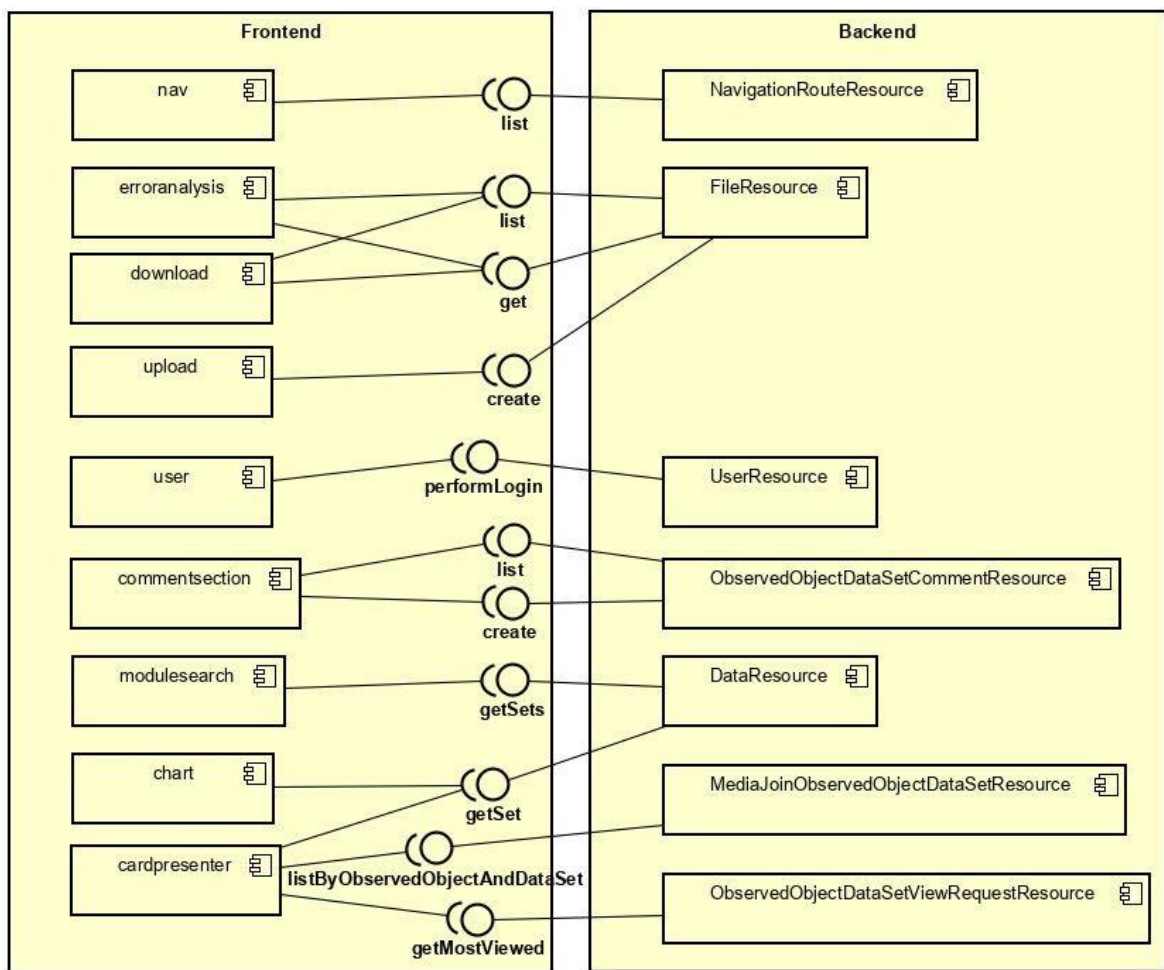


Abbildung 39: Komponentendiagramm

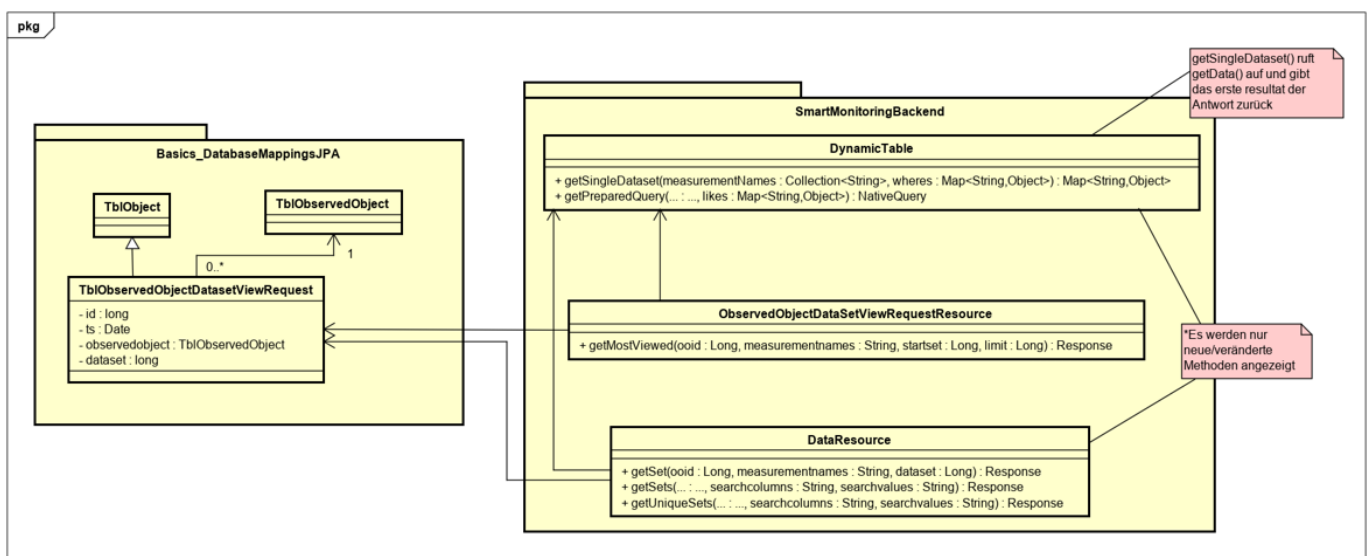


Abbildung 38: Klassendiagramm Suche-View-Request

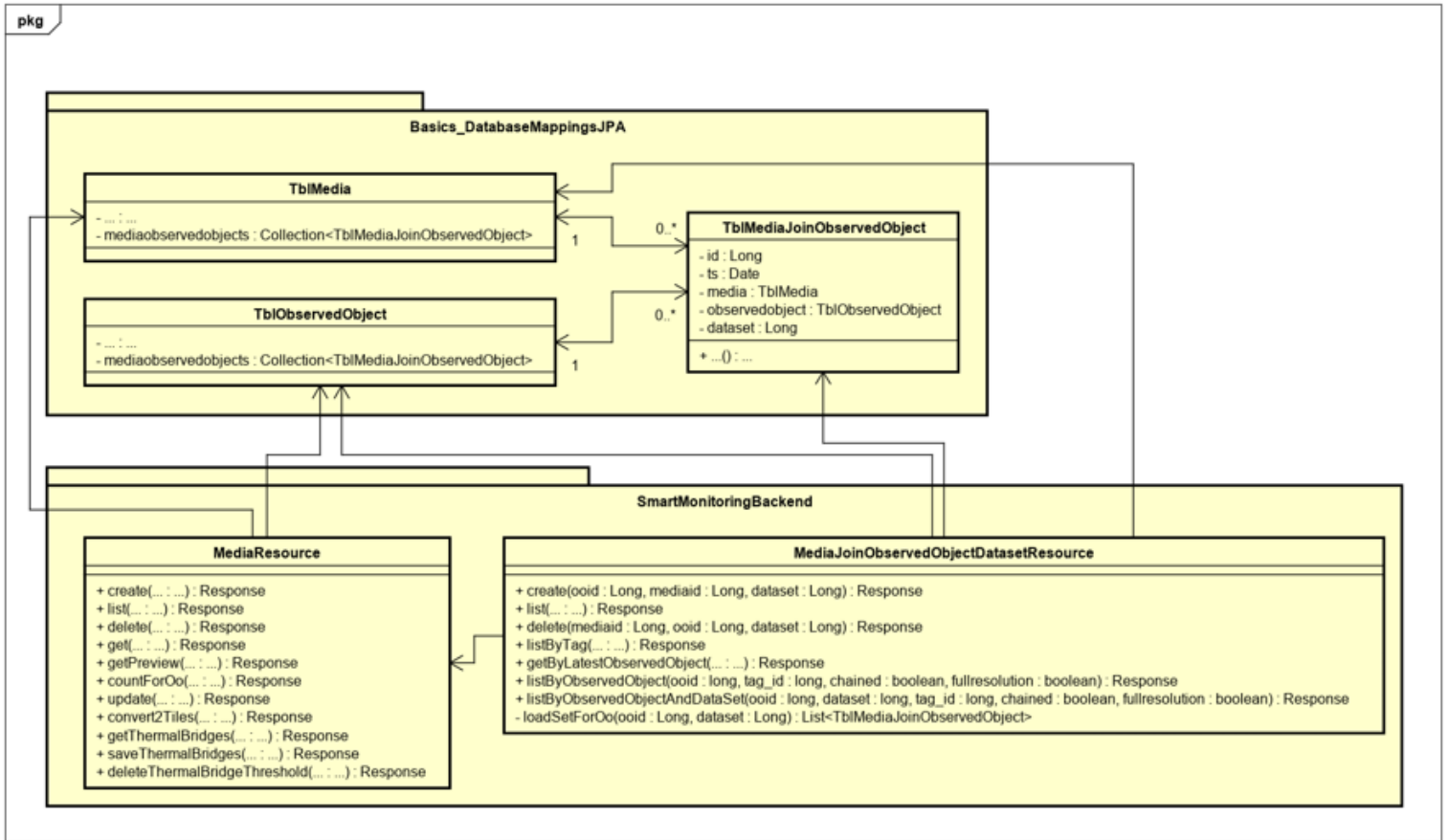


Abbildung 41: Klassendiagramm Medien

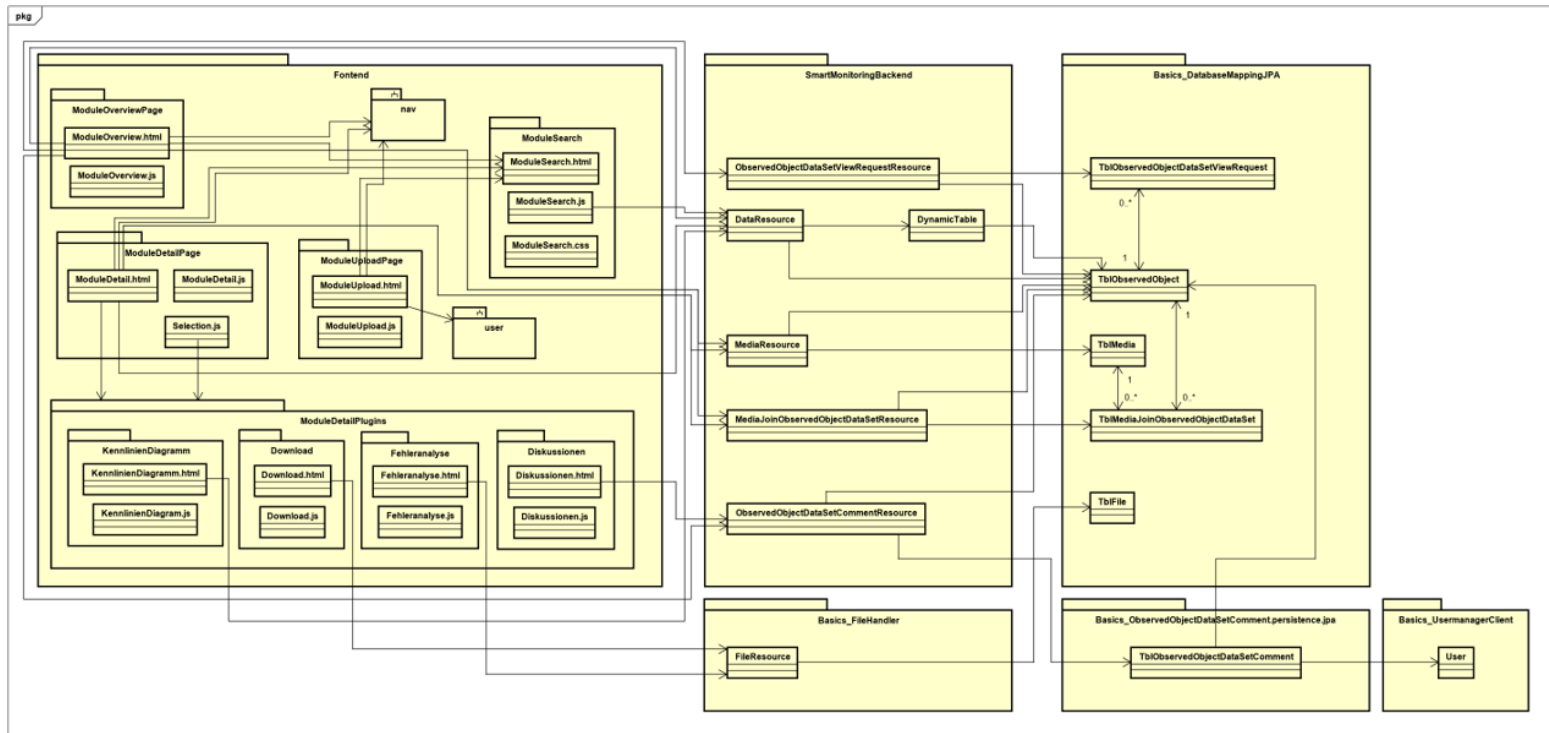


Abbildung 40: Klassendiagramm