PROJEKT-PROTOKOLL

im Studiengang Smart Homes and Assistive Technologies

Lehrveranstaltung Verteilte Systeme

Smart Home Webservice

Ausgeführt von:

DJUKIC Dimitrije, sa18b034

JOVIC Stefan, sa19b003

KURTAJ Fatlum, sa20b034

WINTER Reinhard, sa19b010

WEISS Katharina, sa19b023

Wien, 22.6.2021

Inhaltsverzeichnis

[1 Projektbeschreibung 3](#_Toc75293783)

[1.1 Tools & Technolgoies 3](#_Toc75293784)

[2 Website als Client 4](#_Toc75293785)

[2.1 HTML, CSS und PHP 4](#_Toc75293786)

[2.2 Xampp, Apache, SQL-Datenbank 5](#_Toc75293787)

[3. RESTful Service 6](#_Toc75293788)

[3.1 Licht 6](#_Toc75293789)

[3. 2 Test mit YARC 11](#_Toc75293790)

[3.3 Jalousie 17](#_Toc75293791)

[3.4 Cloud Deployment 21](#_Toc75293792)

[4. SOAP 22](#_Toc75293793)

[4.1 SOAP Webservice 22](#_Toc75293794)

[5 Final Look 26](#_Toc75293795)

[6 Dokumentation und Version Control 29](#_Toc75293796)

[7 Lessons learned 29](#_Toc75293797)

## 1 Projektbeschreibung

Das Abschlussprojekt im SS21 für die Lehrveranstaltung „Verteilte Systeme“ ist ein Smart Home Projekt zu erstellen. Das Projekt soll als Web- oder Mobile App erstellt sein und zudem zwei Web Services zur Verfügung stellen:

* Wetter Web Service (SOAP)
  + Abrufen aktueller Temperatur
  + Abrufen aktueller Windrichtung und -geschwindigkeit
  + Abrufen einer 7 Tages Prognose
* Smart Home Web Service (RESTful)
  + Steurung von Licht
    - Rauf/runter
    - Hinzufügen/Löschen
    - Gruppenschaltunge
  + Steuerung von Jalousie
    - Rauf/Runter
    - Hinzufügen/Löschen
    - Bei Wind > 70 km/h Jalousie rauf

## Tools & Technolgoies

Für das Projekt wurden verschiedene Tools und Technologien benutzt:

* **Betriebssysteme**: Windows 10 & Ubuntu 18.4
* **SOAP**:
  + IDE: IntelliJ
  + Maven
  + SpringBoot
  + Postman
  + AWS
  + Java
* **RESTful**
  + Visual Studio
  + Azure
  + C#
  + YARC
* **Website als Client**
  + Html
  + CSS
  + PHP
  + XAMPP
  + mySQL
  + Eclipse
* **Sonstiges**
  + GitHub

## 2 Website als Client

## HTML, CSS und PHP

Um den Client zu erstellen (Web App), haben wir Visual Studio Code verwendet. Hierfür wurden einfache HTML und PHP-Befehle benutzt, um verschiedene Webseiten zu erstellen und zu verbinden. Für die Optik wurde noch ein CSS File erstellt.

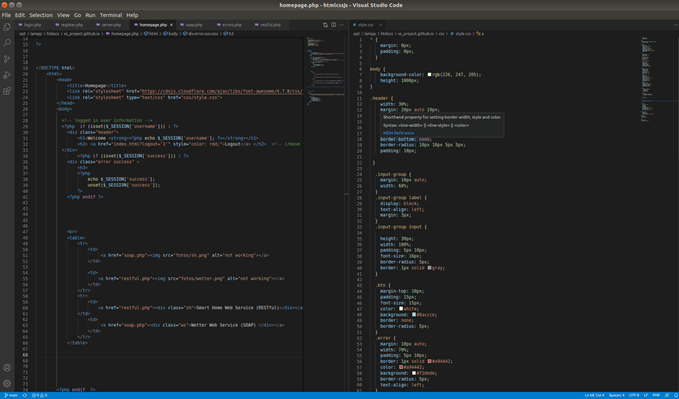


Figure 1 Visual Studio Code

PHP spielt eine große Rolle da es nicht nur für Cookies und Sessions benutzt wird, sondern auch um Seiten zu verlinken, überprüfen, ob das Formular vollständig ausgefüllt ist, Fehler anzuzeigen, verschiedene Kalkulationen und Umwandlungen zu erledigen, die Verbindung zu den Datenbanken sicher zu stellen und uns die Möglichkeit bietet die Daten von einem Web Service anzuzeigen.



Figure 2 PHP Datenbankverbindung

## Xampp, Apache, SQL-Datenbank

XAMPP ermöglicht das Installieren und Konfigurieren des Webservers Apache mit der Datenbank. MariaDB und die PHP-Sprache. XAMPP kann man in Ubuntu mit einem einfachen Command: sudo /opt/lampp/lampp start laufen lassen. Bei Default nutzt die Apache Port 80. Dieser kann bei Bedarf in den config Dateien angepasst werden.

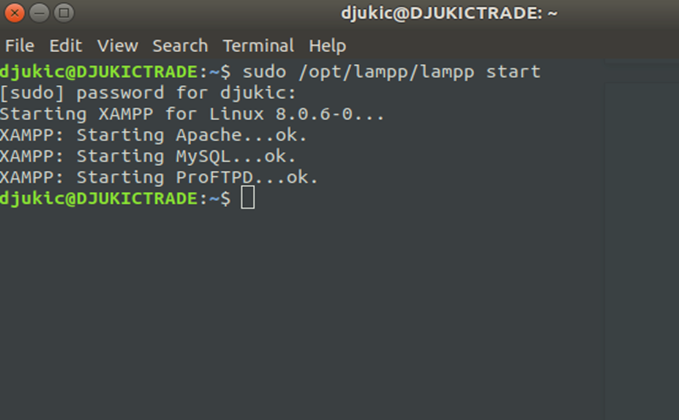


Figure 3 XAMPP starten

Mittels server.php kann eine Verbindung zur Datenbank hergestellt werden und dann im GUI die Umgebung der Datenbank zu verändern, Nutzer hinzuzufügen, Tabellen zu bearbeiten usw. Wir haben hier eine Datenbank erstellt die die Nutzer unserer Website sowie deren Login Daten verwaltet. Wenn auf der Website sich ein neuer Nutzer registriert werden dessen Daten in der Datenbank gespeichert und der Nutzer kann sich anschließend einloggen. Sollte sich ein nicht registrierter Nutzer einloggen wollen folgt eine Fehlermeldung. Zugriff auf die Datenbank bekommt man über folgende URL: *localhost.phpmyadmin*

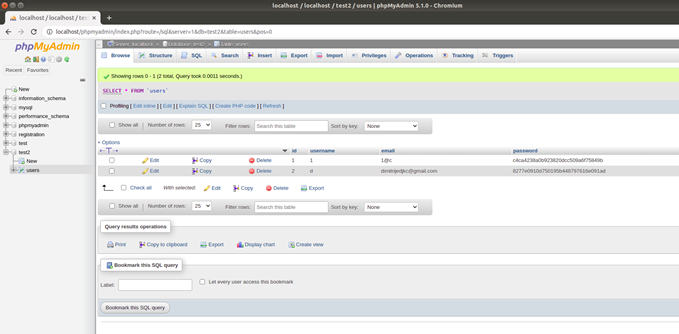


Figure 4 phpMyAdmin

## 3. RESTful Service

## 3.1 Licht

Die Aufgabenstellung war, eine Restful Anwendung zu erstellen, die über einen Web Client abrufbar ist. Der Restful Service wurde in Visual Studio geschrieben. Weiters sollte man die Werte über eine Cloud, wie Microsoft Azure oder Amazon Service, ansprechen lassen.

Da man im Smarthome auch Lichter ansteuern soll, wurde noch die Klasse „Licht“ implementiert mit den Attributen Id, Name, Status und Lichtstärke, welches man mit get und set über den Web Client ansprechen kann. Der Status false oder true gibt an, ob das Licht ein oder aus ist. Die Lichtstärke vermittelt die Lichtintensität und hat einen Zahlenbereich von 0 – 100.

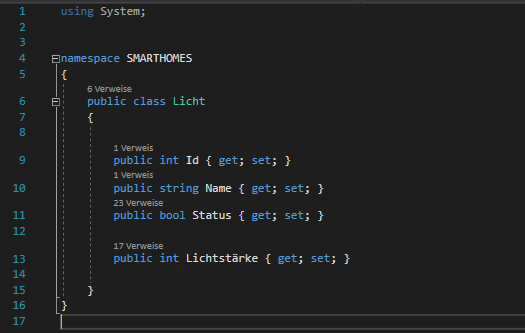
****

Figure 5 Klasse Licht

Beim Lichtcontroller wurden die Namen LichtWohnzimmer, LichtWC,LichtKüche und LichtSchlafzimmer gewählt. In der Zeile 39 wurde die minimale und maximale Anzahl der Objekte bestimmt. Die Lichtstärke und der Status werden hierbei zufällig ausgegeben. Im LichtController wurde noch ein Logger benutzt, der Zufallswerte zum entsprechenden Attribut ausgeben soll.

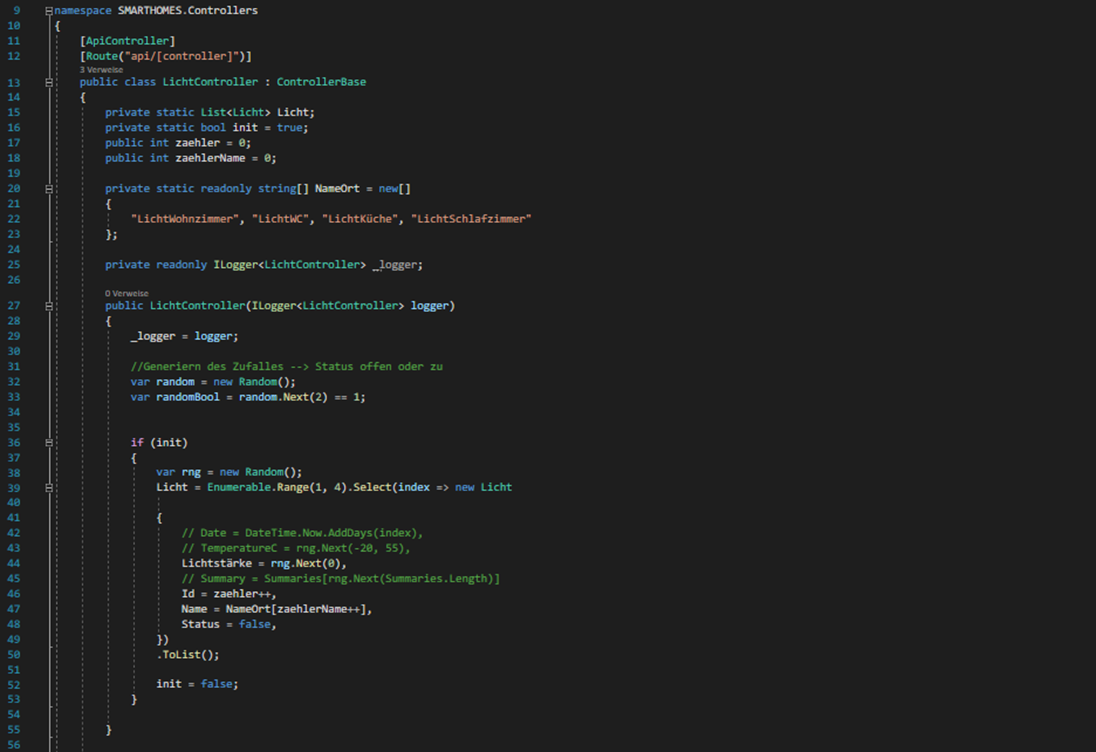
****

Figure 6 Licht Controller

Mit “HttpGet” wird das Objekt Licht ausgegeben. Je nachdem welche ID man ausgeben will, kann mit zum Beispiel localhost:44320/api/Licht/0 den ersten Eintrag ausgeben. Bei einem Post Aufruf wird mit Hilfe des Befehls „Licht.Add(L);“ ein neues Objekt hinzugefügt.

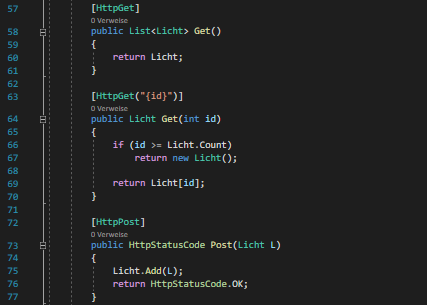
****

Figure 7 Get/Post Abfragen

Zusätzlich wurden noch 4 verschiedene Modus hinzugefügt. Wenn man über YARK ein Put Befehl ausführt und den Link localhost:44320/api/Licht/0 eingibt, dann wird hier der Modus „Lesemodus“ aktiviert.

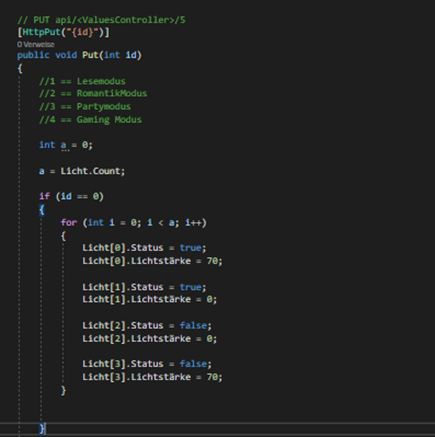
****

Figure 8 Put Abfrage Licht

Um alle Lichter ein – oder auszuschalten wurde in den Zeile 169 bis 182 zwei If-Schleifen hinzugefügt. Gibt man beim PUT Aufruf die ID „111“ ein, so werden alle Lichter eingeschaltet(true). Umgekehrt, wenn man als ID die „999“ eingibt, werden alle Lichter ausgeschaltet(false). Mit Hilfe des unteren Teils kann man auch einzelne Lichter ein- und ausschalten.

****

Figure 9 PUT Abfrage Licht

## 3. 2 Test mit YARC

YARC steht für Yet Another Rest Client. Um das Programm in Visual Studio zu testen, mussten wir uns noch die Chrome Extension Yarc installieren. Mit Hilfe von Yarc konnten wir die Werte mittels Get, Put, Post und Delete ausgeben, hinzufügen,ändern und löschen.

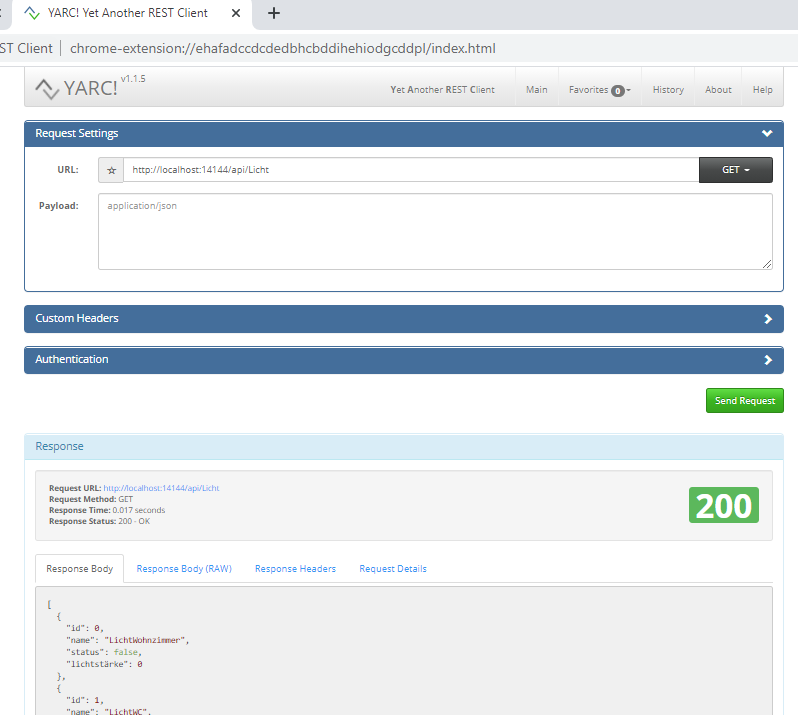


Figure 10 Get Abfrage mit YARC

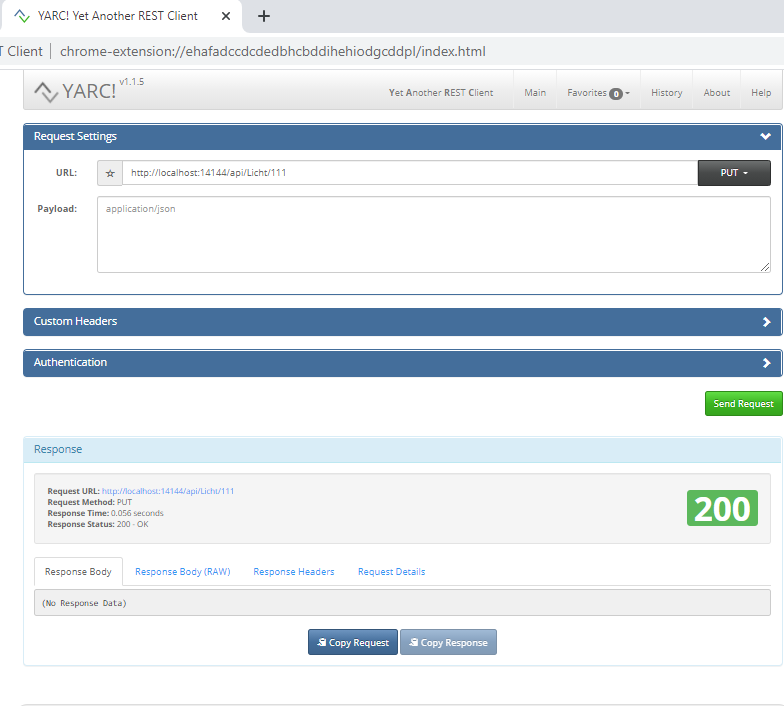
Um alle Lichter einzuschalten muss man, je nachdem wie es programmiert wurde, in Yarc die entsprechende ID eingeben. Wenn man nun als ID die “111” eingibt, so sollten alle Lichter eingeschalten werden.

Figure PUT Befehl in YARC

Ruft man erneut die Get-Methode auf. so wird folgendes angezeigt:



Figure 12 Ergebnis der GET Methode

Möchte man ein Objekt löschen, so ruft man die DELETE Methode auf gibt in der URL über die ID das gewünschte Objekt an.

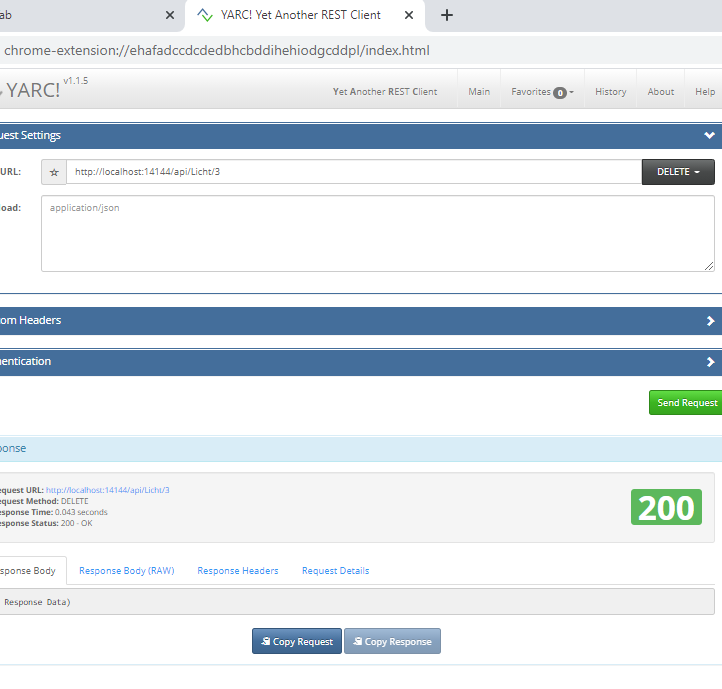


Figure 13 Delete Abfrage

Danach kann man über die POST Methode ein neues Objekt, mit beliebigen Werten, hinzufügen.

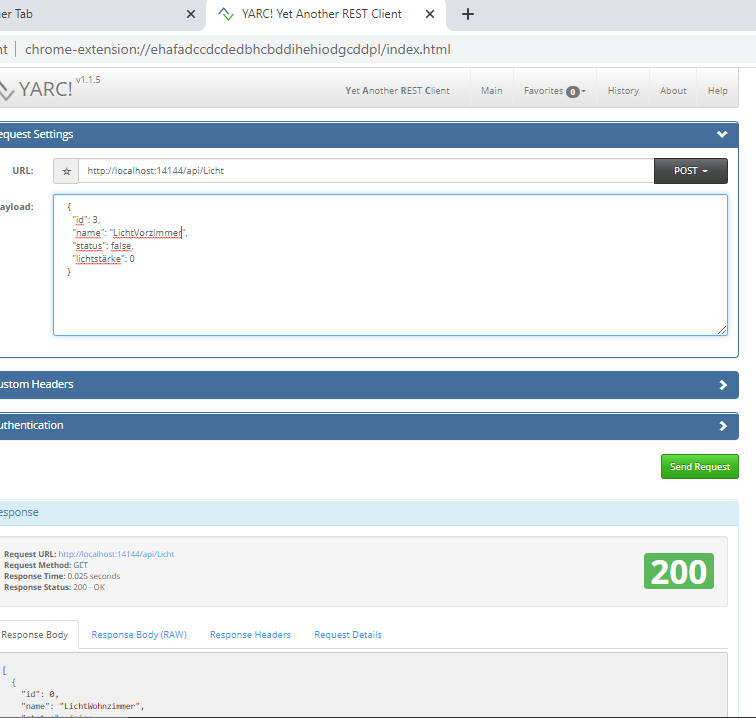


Figure 14 POST Abfrage

Wie in der unteren Abbildung ersichtlich, wurde ein neues Objekt hinzugefügt.

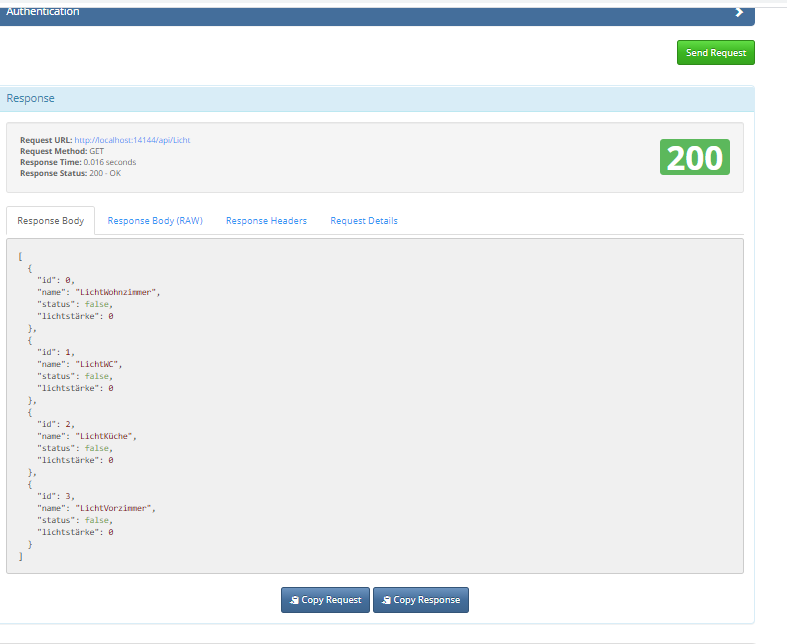
****

Figure 15 GET Abfrage nach POST Methode

## 3.3 Jalousie

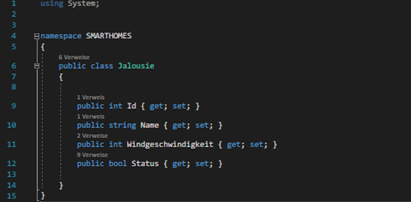


Figure 16 Jalousie Klasse

Für die Jalousie haben wir eine Class erstellt mit den folgenden Attributen: Id, Name, Windgeschwindigkeit und Status. Id gibt den aktuellen Index zurück wie man diesen ansprechen kann bezüglich den einzelnen Get, Post, Put und Delete Request. Der Name gibt den Namen der Jalousie zurück. Die Windgeschwindkeit jeweils eine zufällige Windgeschwindigkeit und der Status, ob die Jalousie unten oder oben ist. Mit den jeweiligen get und set kann man auf die Werte zugreifen.

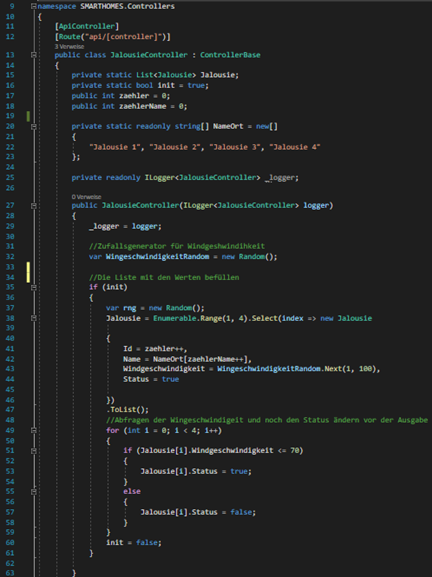


Figure 17 Jalousie Controller

In der Zeile 15-18 hätten wir globale Variablen, die wir für die Funktionalität des Programmes benötigen. Dabei hätten wir in der Zeile 15 die Liste für die Jalousie wo wir jeweilige Einträge haben mit den Attributen. In dieser Liste kann man später Jalousien Hinzufügen, Löschen, Aktualisieren und die Aktuelle Liste Abfragen. Zeile 16 ist eine „bool“ Hilfsvariabe um eine Anweisung abzufragen, nämlich die in der Zeile 35. Zeile 17 und 18 sind ebenso „int“ Hilfsvariablen die wir benutzen um bei der Erzeugung der Liste den jeweiligen Wert hoch zu zählen.

In der Zeile 32 hätten wir die Etablierung der zufälligen Windgeschwindigkeit, die wir im späteren Verlauf benötigen. Zeile 35 bis 47 ist die Befüllung der jeweiligen Attribute, die in der Class vorkommen. Dabei starten wir immer mit 4 Einträgen von Jalousie1 bis Jalousie4. Zusätzlich hätten wir noch in der Zeile 49-59 eine Schleife die, die 4 Einträge durchgeht und bei einer Windgeschwindigkeit >70 den Status der Jalousie von „true“ = offen auf „false“ = zu ändert.

Ist dieser Prozess abgeschlossen so wird der init-Wert in Zeile 60 auf „false“ gesetzt folge dessen springt man aus der Anweisung in Zeile 35 raus.

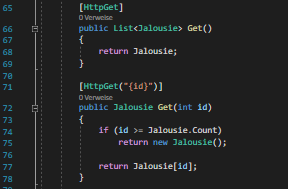


Figure 18 GET Methode Jalousie

Mit „HttpGet“ wird das Objekt Jalousie ausgegeben, indem man alle Einträge der Jalousie sieht. Je nachdem welche ID man ausgeben will „HttpGet(“{id}“), kann man mit localhost:44320/api/Jalousie/0 den ersten Eintrag ausgeben, dies wird mittels Zeile 71 gewährleistet, da der jeweilige Eintrag aus der Liste ausgewählt wird.

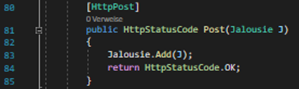


Figure 19 POST Methode Jalousie

Mittels „HttpPost“ kann man zusätzliche Einträge in der Jalousie hinzufügen, wie man mittels des Befehles in der Zeile 83 sieht.

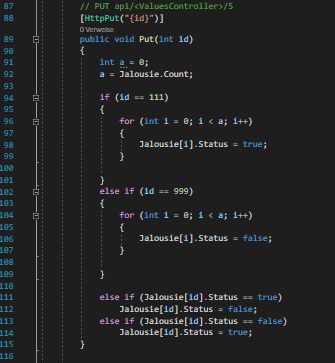


Figure 20 PUT Methode Jalousie

Mittels „HttpPut(“{id}“)“ kann der Status der ausgewählten Jalousie verändert werden. Möchte man von allen Jalousien den Status auf offen ändern so muss man beim Put die Zahl „111“ hinzufügen wie zum Beispiel: localhost:44320/api/Jalousie/111. Dies wird in der Zeile 94-101 abgefragt. Ebenso funktioniert zum Schließen aller Jalousie mit „999“ dies wird in der Zeile 102-109 abgefragt. Falls nur ein Eintrag geändert werden muss, erfolgt dies in der Zeile 111-114 wo der Status der erwähnten Jalousie geändert wird.

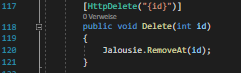


Figure 21 DELETE Jalousie

Mittels „HttpDelete(“{id}“)“ kann der jeweils ausgewählte Eintrag komplett aus der Liste raus gelöscht werden.

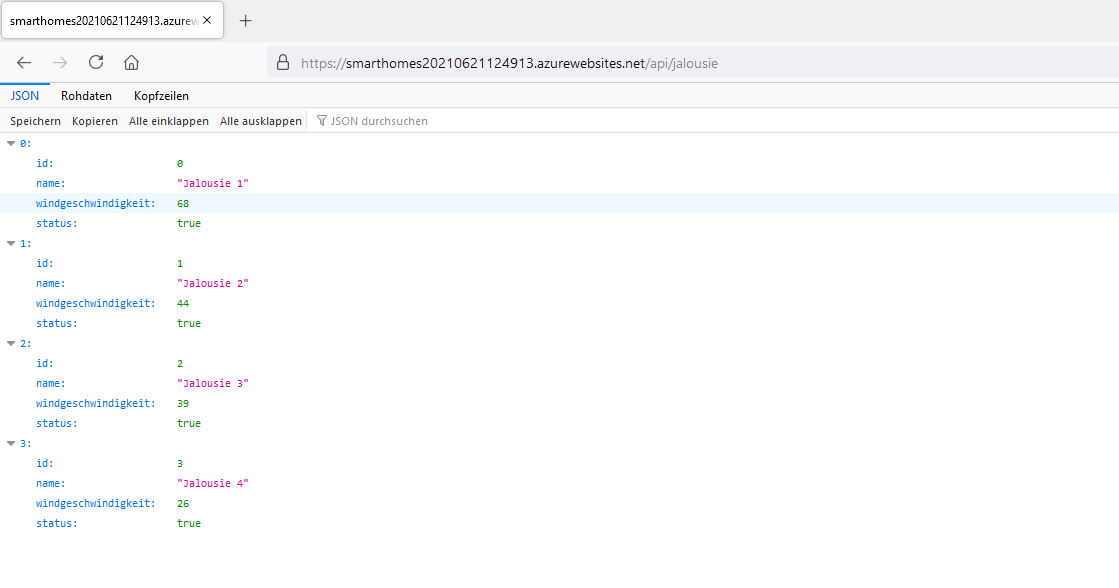


Figure 22 Jalousieobjekte

## 3.4 Cloud Deployment

Um sich mit der Cloud Microsoft Azure verbinden zu können, musste man sich zuerst ein Account in Microsoft erstellen. Anschließend wurde in Visual Studio die notwendigen Konfigurationen eingestellt

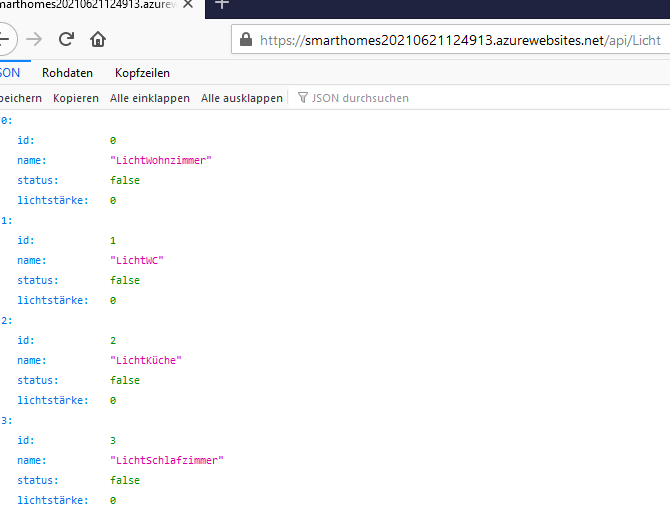


Figure 23 Lichtobjekte aus der Cloud

## 4. SOAP

Die Anforderung für den SOAP Client waren wie folgt:

* momentane Temperatur anzeigen
* Windrichtung und Windgeschwidigkeit
* Wetter Prognose für die nächsten 7 Tage

## 4.1 SOAP Webservice

Wir haben uns entschieden dafür in IntelliJ ein Spring Boot Projekt an zu legen das mittels Maven verwaltet wird. Um Den Service zum Laufen zu bringen müssen als erstes im pom.xml file die Dependencies und Properties korrekt angegeben werden. Das bedeutet vor allem welche Java Version, in unserem Fall Java 8, welches Framework und die Dependencies für das WSDL File.

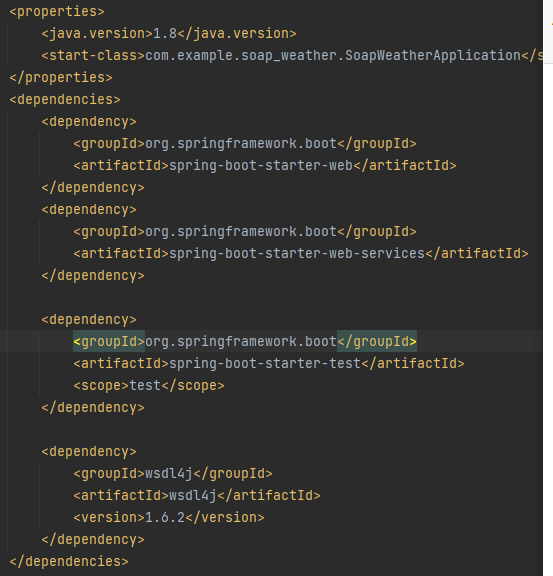
****

Figure 24 Dependencies und Properties für Spring

Im nächsten Schritt wird das Konstrukt für den Soap Service in ein XML File geschrieben, in unserem Fall weather.xml. Dabei haben wir die Struktur für die Request und Response Funktionen vorgegeben. Wir haben uns dazu entschieden das eine Parameter Übergabe nicht nötig ist für diesen Service, weshalb dir Requests leer sind. Der Response für die momentane Temperatur gibt einen float Wert zurück, der Response für den Wind ein Objekt mit einem String für die Richtung und einen float für die Geschwindigkeit und der Response für die Wetterprognose gibt eine Liste von 7 Objekten zurück die einen float für die Temperatur einen String für die Windrichtung und einen float Wert für die Windgeschwindigkeit enthalten. Aus diesem XML File generiert dann Maven die nötigen Java Interfaces. Dafür wird im folgenden Interface clean and install ausgewählt, wobei prinzipiell, wenn nicht schon von Maven erstellte Dateien vorhanden sind auch nur install reichen würde.

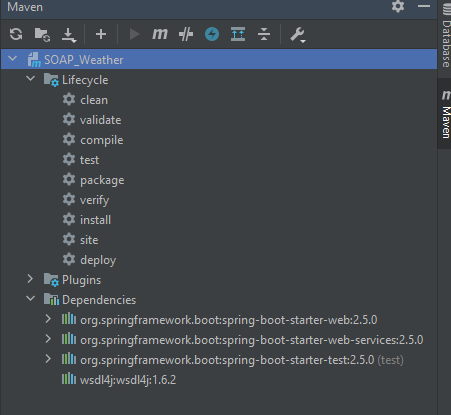
****

Figure 25 Maven Menü

Im nächsten Schritt müssen die Java Klassen implementiert werden die unsere Funktionen beinhalten. Die Temperatur und Geschwindigkeit wird Random generiert und die Windrichtung auch zufällig aus einem Array mit 8 Himmelsrichtungen ausgewählt. Danach muss eigentlich nur noch die Klasse für den Endpoint geschrieben werden. In dieser wird definiert wie die einzelnen Funktionen aufgerufen werden, also über welche URL. Das fertige Projekt mit allen nötigen Klassen sieht dann wie folgt aus:

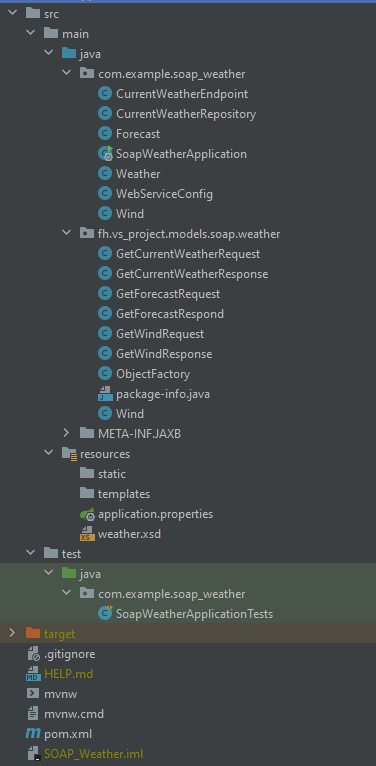
****

Figure 26 Projektstruktur SOAP

Als nächstes kann der Service gestartet werden:

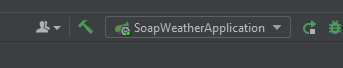
****

Figure 27 Spring builden, compilieren, starten

Wenn alles klappt startet der Spring Boot Server und der SOAP Service kann aufgerufen werden. Im Terminal sieht ein erfolgreicher Start dann wie folgt aus:

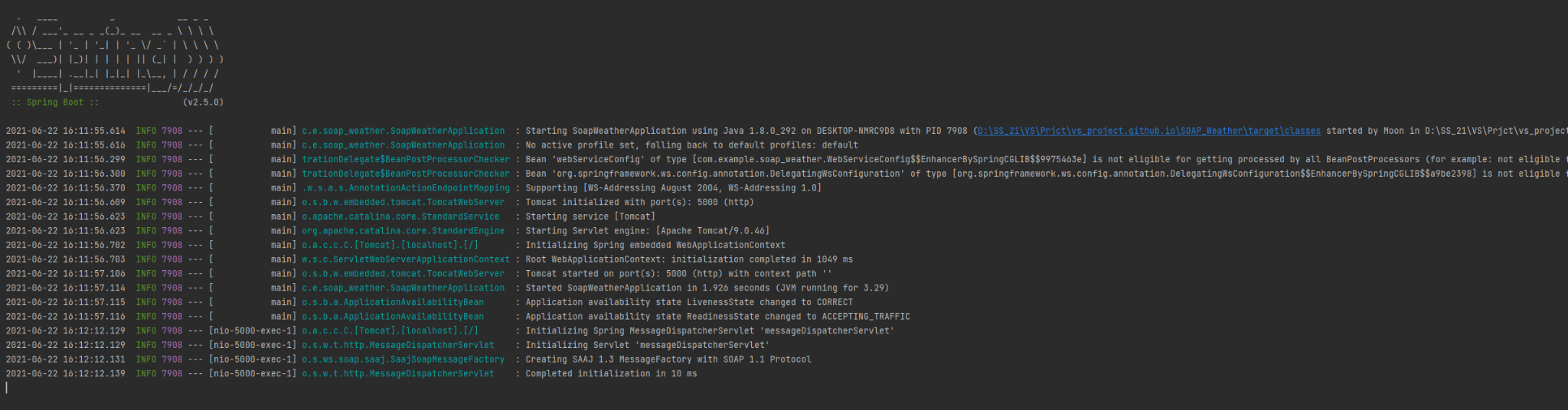
****

Figure 28 Konsolenausgabe Spring

Hier werden dann auch Errors angezeigt und auch weiter Informationen wenn ein Request oder ein Respond geschickt wird. Zum Testen des SOAP Services haben wir uns dazu entschieden Postman zu verwenden. Hierbei ist noch wichtig bei headers text/xml auszuwählen. Im Body kann dann der SOAP Request eingegeben werden und unten wird bei Erfolg die SOAP Response im xml Format angezeigt.

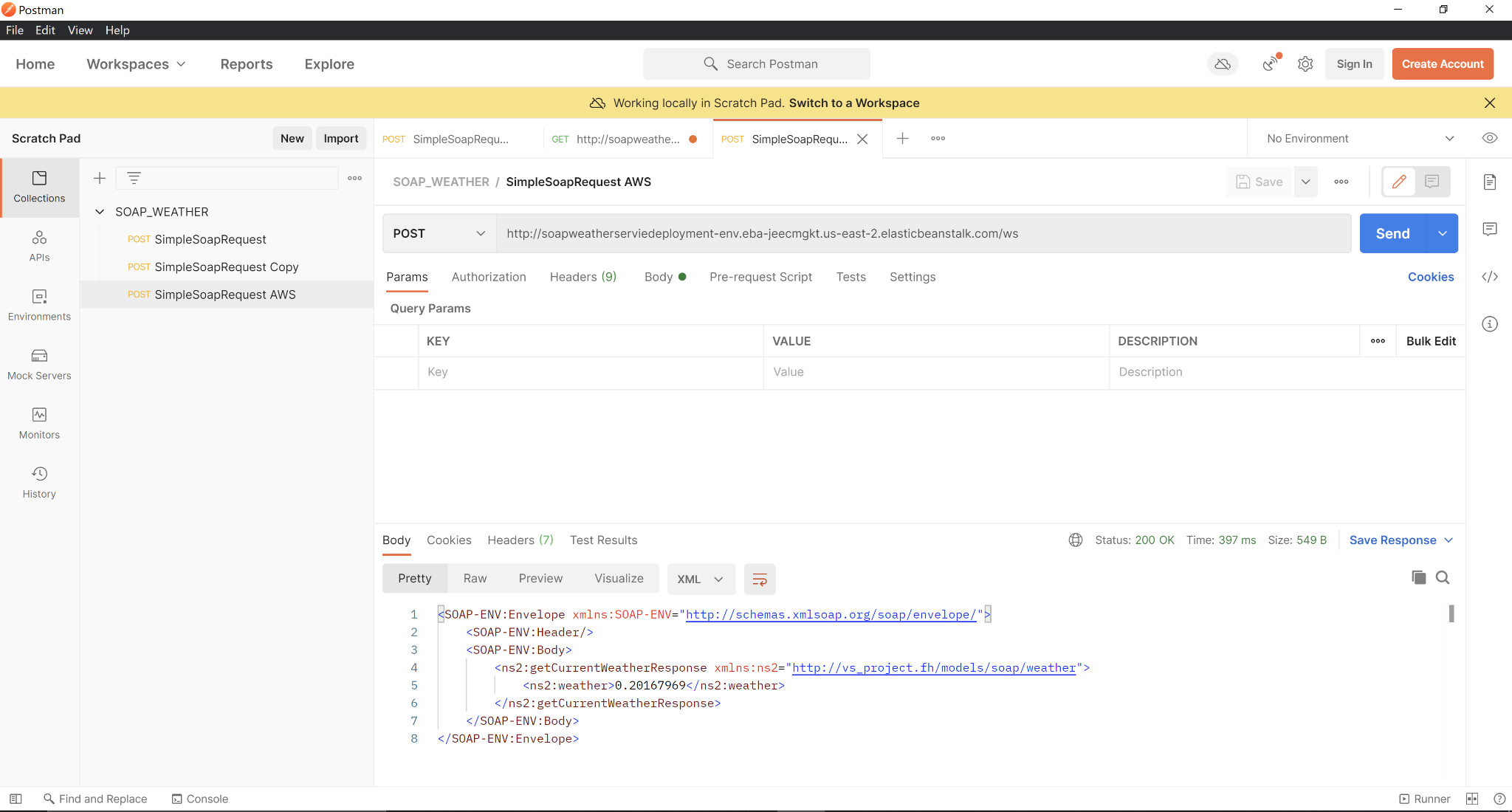


Figure 29 Postman Test

## 4.2 Cloud Deployment

Wir haben uns dazu entschieden den SOAP Service über Amazon Web Services (AWS) bereit zu stellen. Hierfür haben wir erstmal bei den application properties den Server Port auf 5000 umgestellt.

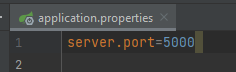
****

Figure Port für Cloud ändern

Im AWS Browser GUI haben wir dann eine Elastic Beanstalk Anwendung erstellt. Hier wurde dann die .jar Datei der Spring Anwendung hochgeladen, den Rest erledigt AWS. Über die angezeigte URL kann dann die Anwendung in unserem Web Client aufgerufen werden.

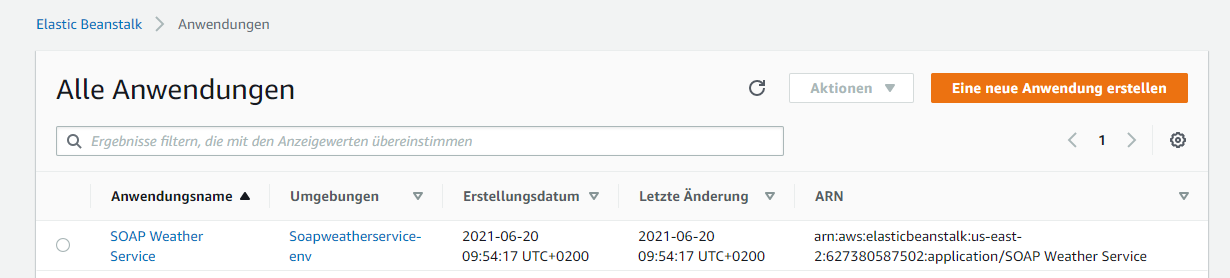
****

Figure Elastic Beanstalk Anwendung

## 5 Final Look

Auf der fertigen Website muss sich der User erstmal registrieren, danach kann er sich einloggen.

****

Figure 32 Login Maske

Nach dem Einloggen kommt man auf die Startseite auf der zwischen dem Smart Home Service und dem SOAP Service gewählt werden kann.

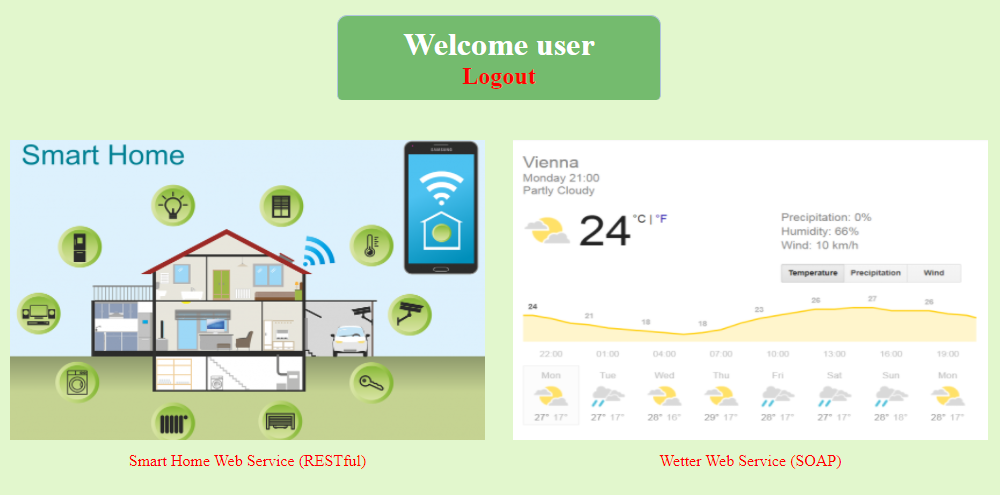
****

Figure 33 Homepage

Auf der Rest Seite ist nochmal die Auswahl zwischen Licht und Jalousie:

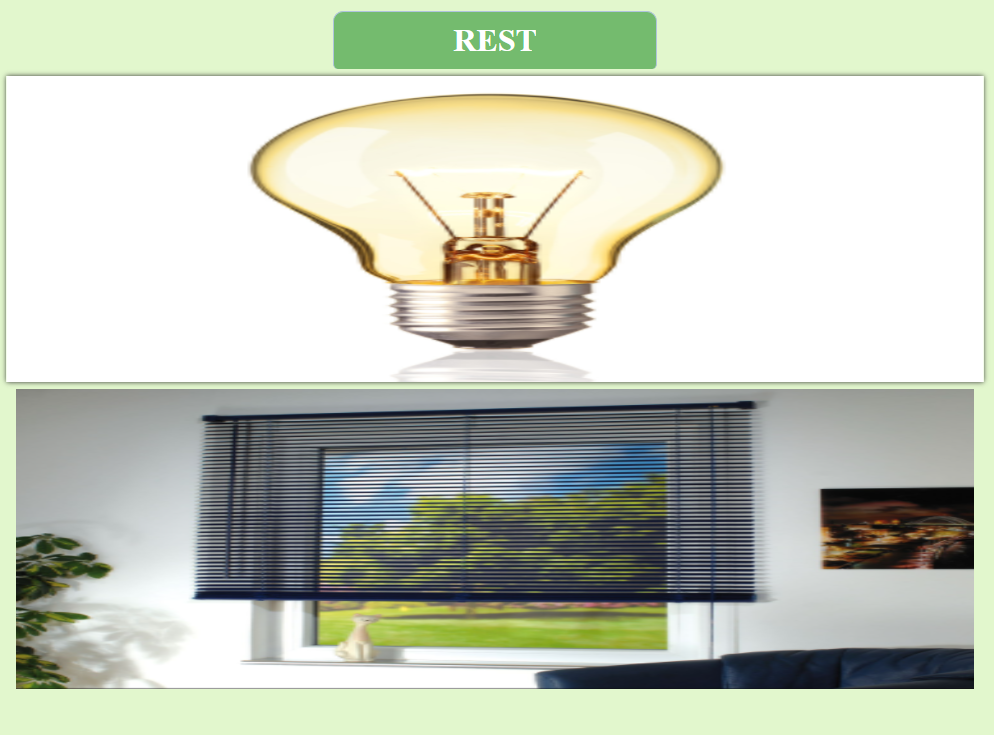
****

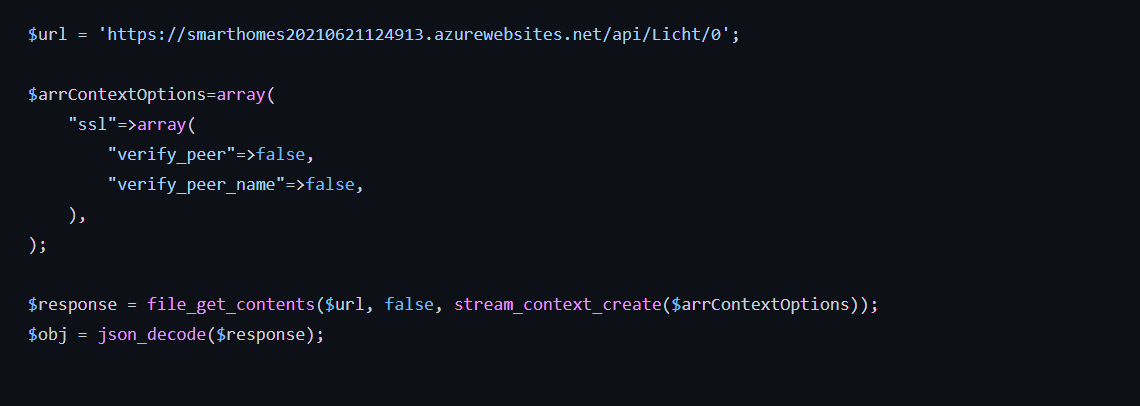
Figure 34 REST Service

Ein Licht GET Request liefert folgendes Ergebnis, von der Cloud abgerufen:

****

Figure 35 Licht Request

Für die Implementierung musste hier die variablen für ssl „verify\_peer“ und „verify\_peer\_name“ auf false gesetzt werden. Dies könnte auch über ein gültiges Zertifikat gelöst werden, aus Zeitgründen haben wir uns aber für die diese Variante entschieden. Die Antwort wird in response gespeichert und dann aus JSON umgewandelt um das Objekt auf der Website darstellen zu können.



Der SOAP Service ruft gleichzeitig alle 3 geforderten Request auf und zeigt sie an:

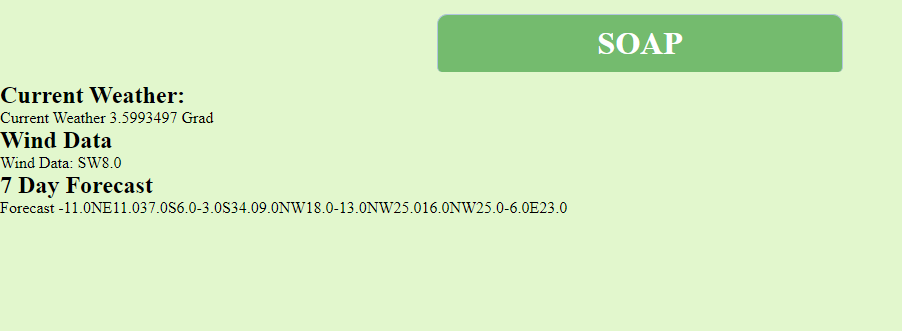
****

Figure 36 SOAP Request

Aus Zeitgründen konnte das leider nicht mehr schöner dargestellt und alle REST Funktionen auf der Clientseite implementiert werden.

Der SOAP Request wurde mittels eines curl Request im Web Client implementiert um das SSL/Https Problem zu umgehen. Hierbei wurden die SSL\_VERIFYHOST und SSL\_VERIFYPEER auf false gesetzt.

Der SOPA Requet wird im xml Format als String übergeben, wie in den Postman Test als String übergeben. Die Antwort wird dann schlussendlich in Data übergeben und data auf der Website ausgegeben.



Figure Web Client SOAP Aufruf

## 6 Dokumentation und Version Control

Für die Zusammenarbeit haben wir auf einem GitHub Account ein neues Projekt angelegt und die anderen als Contributer hinzugefügt. Mittels Pull und Push Request konnte der Fortschritt mit den anderen geteilt werden und das Projekt dezentral verwaltet werden. Geholfen hat hierbei auch die GitHub Desktop App wobei auch ein Teammitglied mit der Konsole gearbeitet hat. Für das Protokoll haben wir unsere Notizen erstmal über Google Docs miteinander geteilt und erst am Schluss daraus ein formatiertes Word Dokument gemacht.

## 7 Lessons learned

**Website**

User Registrierung Datenbanken zu erstellen und mit PHP zu verbinden

**SOAP:**

Es braucht mehr als nur eine IDE um einen SOAP Service zu implementieren, auch muss darauf geachtet werden bei so einem Projekt, bei dem 2 Services und ein Web Client auf einem Rechner laufen könnten das unterschiedliche Ports eingestellt werden. Bei der Recherche im Internet fällt oft auf das statt SOAP ein RESTful Service vorgeschlagen wird. Es scheint das ein RESTful Service Anfängerfreundlicher ist und auch deutlich mehr Support, Librarys und Frameworks dazu zu finden sind als zu SOAP. Auch muss sehr darauf geachtet werden in welcher Version Dinge verwendet werden, oftmals sind Dinge aus diversen Tutorials nicht anwendbar, weil sie sich auf ältere oder neuere Versionen beziehen. Ebenfalls ein Problem stellt hierbei die aktuelle PHP-Version dar. Die Anleitungen, die wir gefunden haben bezogen sich hauptsächlich auf http Request, gebraucht haben wir allerdings einen https Request was uns vor die Herausforderung gestellt hat wie wir das mit dem Zertifikat machen, damit wir keinen ssl Fehler geworfen bekommen. Wir haben dieses Problem schlussendlich umgangen, indem wir entsprechende Variablen gesetzt haben und damit suggeriert haben das der Zugriff passt auch wenn kein Zertifikat vorliegt.

**GIT:**

Merge Konflikts lösen, Probleme lösen, die entstehen wenn mehrere Leute an einem echten Projekt remote arbeiten.

Abbildungsverzeichnis

[Figure 1 Visual Studio Code 4](#_Toc75294517)

[Figure 2 PHP Datenbankverbindung 4](#_Toc75294518)

[Figure 3 XAMPP starten 5](#_Toc75294519)

[Figure 4 phpMyAdmin 6](#_Toc75294520)

[Figure 5 Klasse Licht 7](#_Toc75294521)

[Figure 6 Licht Controller 7](#_Toc75294522)

[Figure 7 Get/Post Abfragen 8](#_Toc75294523)

[Figure 8 Put Abfrage Licht 9](#_Toc75294524)

[Figure 9 PUT Abfrage Licht 9](#_Toc75294525)

[Figure 10 Get Abfrage mit YARC 10](#_Toc75294526)

[Figure 11 PUT Befehl in YARC 11](file:///C:\Users\Katharina\Downloads\Verteilte_Systeme_Projektprotokoll.docx#_Toc75294527)

[Figure 12 Ergebnis der GET Methode 12](#_Toc75294528)

[Figure 13 Delete Abfrage 13](#_Toc75294529)

[Figure 14 POST Abfrage 13](#_Toc75294530)

[Figure 15 GET Abfrage nach POST Methode 14](#_Toc75294531)

[Figure 16 Jalousie Klasse 14](#_Toc75294532)

[Figure 17 Jalousie Controller 15](#_Toc75294533)

[Figure 18 GET Methode Jalousie 16](#_Toc75294534)

[Figure 19 POST Methode Jalousie 16](#_Toc75294535)

[Figure 20 PUT Methode Jalousie 17](#_Toc75294536)

[Figure 21 DELETE Jalousie 17](#_Toc75294537)

[Figure 22 Jalousieobjekte 18](#_Toc75294538)

[Figure 23 Lichtobjekte aus der Cloud 18](#_Toc75294539)

[Figure 24 Dependencies und Properties für Spring 19](#_Toc75294540)

[Figure 25 Maven Menü 20](#_Toc75294541)

[Figure 26 Projektstruktur SOAP 21](#_Toc75294542)

[Figure 27 Spring builden, compilieren, starten 21](#_Toc75294543)

[Figure 28 Konsolenausgabe Spring 22](#_Toc75294544)

[Figure 29 Postman Test 22](#_Toc75294545)

[Figure 30 Login Maske 23](#_Toc75294546)

[Figure 31 Homepage 23](#_Toc75294547)

[Figure 32 REST Service 24](#_Toc75294548)

[Figure 33 Licht Request 24](#_Toc75294549)

[Figure 34 SOAP Request 25](#_Toc75294550)