WBA-Modellierung

Überblick der Komponenten

Komponente	Spezifikation	Technologie	Sprache
Middleware	Message Queue	RabbitMQ	Erlang
Server	Anwendungsserver	NodeJS	Javascript
Client (Patient, Personal)	mobiles Endgerät, Tablet	Android OS	Java
Client (Arzt)	lokaler Arbeitsrechner	Internet-Browser	Javascript
Datenbank	relationales DBMS	Oracle MySQL	MySQL
Webservice	Medikamentenkontrolle	NodeJS	Javascript

Überblick der Anwendungslogik

Komponente	Anwendungslogik
NodeJS-Server	- **Medikamentenkontrolle** - **Benachrichtigungen** - **Verwaltung der Datenanfragen (get, post)**
Android-Client	- **Notifications** - **Erstellen der Medikationspläne**

Ressourcen:

Ressource	Methoden
/patient:id	**get, post**
/verordnung	**post**
/login	**post**
/medikamente	**get**
/checkInteraction	**post**

Die Ressourcen werden hauptsächlich von dem Arzt-Client aus aufgerufen. Aufgrund von Komplikationen bei der Anbindung von RabbitMQ an einem Javascript-Client, wurde an dieser Stelle auf das http-Protokoll zurückgegriffen werden, um die notwendigen Daten für den Arzt-Client zu beschaffen. Dabei soll der Arzt-Client mit dem NodeJS-Server kommunizieren.

Queues:

Bezeichnung	Beschreibung	Beispiel	Routing-Keys
patientID	Jeder Patient erhält seine eigene Queue. Der Name der Queue setzt sich aus dem **Benutzertyp** “patient” und der **ID** “patient_id” aus der MySQL Datenbank zusammen. Die Queue wird verwendet, um Benachrichtigungen über den Patienten mit der in dem	**patient1** **patient2** **patient3** ...	**stationID.patientID**

	Routing Key angegebenen **patientID** zu erhalten		
patientID.get	Diese Queue wird benötigt, um ein GET auf den Server auszuführen. Die Queue nimmt dabei die Daten des Servers entgegen. (siehe RPC mit RabbitMQ)		**patientID**.get
personalID	Jedes Personal erhält seine eigene Queue. Der Name der Queue setzt sich aus dem **Benutzertyp** “personal” und der **ID** “personal_id” aus der MySQL Datenbank zusammen. Die Queue wird verwendet, um Benachrichtigungen über den Patienten mit der in dem Routing Key angegebenen **patientID** zu erhalten.	**personal1** **personal2** **personal3** ...	**stationID.patientID**
personalID.get	Diese Queue wird benötigt, um ein GET auf den Server auszuführen. Die Queue nimmt dabei die Daten des Servers entgegen. (siehe RPC mit RabbitMQ)		**personalID**.get
rpc_queue	Diese Queue wird benötigt, um eine synchrone Kommunikation über	Queue ist einzigartig	get, post

	RabbitMQ zu realisieren. Der NodeJS-Server verwaltet die GET-Befehle der Clients. Der Consumer dieser Queue ist daher der NodeJS-Server. Um Daten die vom Client angeforderten Daten zu beschaffen, müssen bestimmte Methoden auf dem NodeJS-Server aufgerufen werden. Dafür wird ein RPC über RabbitMQ verwendet. (siehe Abbildung 1)		

Routing-Keys (Topics)

Bezeichnung	Nachricht	Beschreibung

get	{
	“**Absender**”,
	“**Methode**”,
	“**Parameter**”
	}
		Für die Verwaltung der GET-Befehle benötigt der NodeJS-Server Informationen von den Clients, die eine Nachricht auf den Routing-Key **get** gesendet haben. (siehe Abbildung 1)
		
		Absender:
		Als Absender wird der Queuename des Benutzers angegeben, der über den Client, die Nachricht gesendet hat.

		Methode: An dieser Stelle wird der Methodenname der Methode übergeben, die ausgeführt werden soll. **Parameter**: Hier werden die Parameter übergeben, die für den Methoden Aufruf verwendet werden Die Daten werden im JSON-Format der Nachricht übergeben.

post	{
	“**absender**”
	,
	“**methode**”,
	“**daten**”
	}
		Für die Verwaltung der POST-Befehle benötigt der NodeJS-Server Informationen von den Clients, die eine Nachricht auf den Routing-Key **post** gesendet haben.
		(siehe Abbildung 1)
		
		Absender:
		Als Absender wird der Queuename des Benutzers angegeben, der über den Client, die Nachricht gesendet hat.
		
		Methode:
		An dieser Stelle wird der Methodenname der Methode übergeben, die ausgeführt werden soll.
		
		Daten:
		Hier werden die Daten übergeben, die für den Methodenaufruf verwendet werden, um

		die Daten in die Datenbank zu schreiben. Die Daten werden im JSON-Format der Nachricht übergeben.

stationID.patientID	{
	“**station_id**”,
	“**patient_id**”,
	“**beschreibung**”
	}
		Dieser Routing-Key wird für das abonnieren von Nachrichten auf Patienten verwendet. Der Key setzt sich aus der station_id und der patient_id zusammen. Die IDs beziehen sich auf die jeweiligen IDs aus den Tabellen “Station” und “Patient” der MySQL-Datenbank (siehe ER-Modell).
		
		Die Abbildung 2 zeigt ein Beispiel für das abonnieren von Patienten.

RPC mit RabbitMQ

In dem unten aufgeführten Schema eines RPC mit RabbitMQ wird die Methode “getPatient” mit dem Parameter “12” auf dem NodeJS-Server aufgerufen. Der Client sendet dazu einen Request mit dem Routing-Key “get” auf den Exchange “amq.topic”. Der Exchange leitet die Nachricht weiter zur Queue “rpc_queue”, die auf Nachrichten mit dem Routing-Key “get” und “post” wartet. Der Server ist auf mit der Queue “rpc_queue” verbunden und kann daher den Request empfangen und verarbeiten. Dazu wird mit Hilfe der mit gegebenen Nachrichten-Struktur, die Methode “getPatient” aufgerufen. Nach der Datenbeschaffung seitens des Servers werden die angeforderten Daten zurück an den Absender geschickt. Dazu wird der angegebene Queue-Name in der Nachricht des Requests (“personal1”) verwendet, um die Daten an den Client weiterzuleiten.

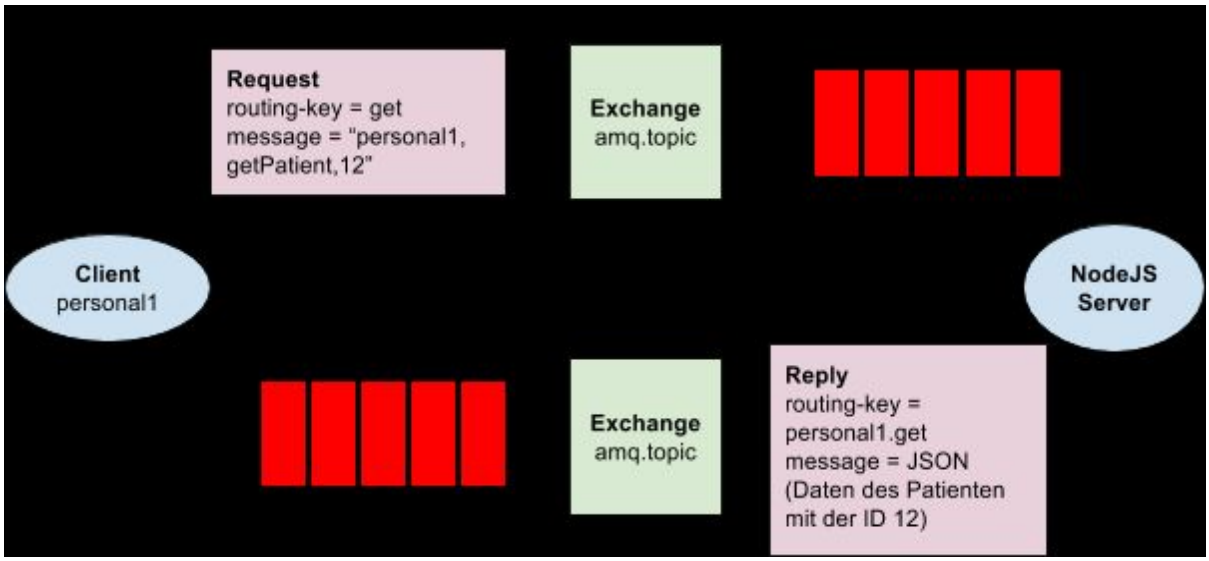

Abbildung 1

Auf dem NodeJS Server müssen für die Realisierung des RPC Methoden implementiert werden, die Daten des Patienten und seiner Verordnungen aus der Datenbank holen. Aufgrund der komplexen großen Menge an Daten wird hierfür das JSON Format verwendet, um einzelne Datenpackete als in ein gesamtes Packet an den Client zu versenden.

JSON Parser

Auf den Android-Clients müssen die Datenpackete, die als Byte-Array vorliegen in ein String umgewandelt werden und anschließend als JSON-Object geparsed werden. Für das parsen eines Strings zum JSON-Object wird die Google Bibliothek "com.google.gson"[1] verwendet. Die Umwandlung kann wie folgt im Android-Client implementiert werden:

```
mParser = new JsonParser();
// message beinhaltet unsere Nachricht aus RabbitMQ als byte[]
String raw= new String(message);
JsonElement obj = mParser.parse(raw);
JsonObject jsonObj = obj.getAsJsonObject();

// Ein Element kann nun mit dem entsprechenden Namen des Elements im JSON entnommen werden
String medikament = jsonObj.get("medikament").toString();
```

Medikationsplan

Die Verordnungen müssen chronologisch nach Einnahmezeit angeordnet werden, bevor sie dargestellt werden können. Die Darstellung der Liste muss anhand der Systemzeit angepasst werden, sodass stets die aktuell zu verabreichenden Verordnung angezeigt werden. Des weiteren sollen nur die Verordnungen angezeigt werden, die für den aktuellen Wochentag relevant sind.

Java-Modelle

Für das Filtern und Sortieren der Verordnungsliste sollen zunächst in Java Modell-Klassen der Daten modelliert werden, da die Verarbeitung in Java dadurch erleichtert wird. Die Daten die vom Server angefordert wurden, können dann den Attributen der instanzierten Objekten der Klassen übergeben werden. Dafür müssen die Daten vom Server ins JSON-Format umgewandelt werden.

Der folgende Java-Code soll eine Implementation der Modelle darstellen:

[1] https://google-gson.googlecode.com/svn/trunk/gson/docs/javadocs/com/google/gson/JsonElement.html
Sichtung: 10.01.2016 18:29

Medikationsplan

```
public class Medikationsplan {
	public int station_id;
	public List<Verordnung> verordnungen;

	public Medikationsplan(JsonArray verordnungen){
		for (int i = 0; i < verordnungen.size(); i++) {
			JsonObject verordnung = verordnungen.get(i).getAsJsonObject();
			this.verordnungen = new ArrayList<Verordnung>();
			this.verordnungen.add(new Verordnung(verordnung));
		}
	}
}
```

Verordnung

```
public class Verordnung {
	public int patient_id;
	public int station_id;
	public int verordnung_id;
	public int zimmer;
	public String beginn;
	public String ende;
	public String patientName;
	public String patientVorname;
	public List<Applikationszeit> applikationszeit;

	public Verordnung(JsonObject verordnung){
		this.patient_id = verordnung.get("patient_id").getAsInt();
		this.station_id = verordnung.get("station_id").getAsInt();
		this.verordnung_id = verordnung.get("verordnung_id").getAsInt();
		this.beginn = verordnung.get("beginn").getAsString();
		this.ende = verordnung.get("ende").getAsString();
		this.patientName = verordnung.get("name").getAsString();
		this.patientVorname = verordnung.get("vorname").getAsString();
		this.zimmer = verordnung.get("zimmer").getAsInt();
```

```
                JsonParser mParser = new JsonParser();
                JsonElement obj = mParser.parse(verordnung.get("applikationszeitpunkt").getAsString());
                JsonArray applikationszeiten = obj.getAsJsonArray();
                this.applikationszeit = new ArrayList<Applikationszeit>();
                for (int i = 0; i < applikationszeiten.size(); i++) {
                        this.applikationszeit.add(new
Applikationszeit(applikationszeiten.get(i).getAsJsonObject()));
                }
        }
}
```

Applikationszeit

```
public class Applikationszeit {
        public String tag;
        public List<String> zeiten;

        public Applikationszeit(JsonObject applikationszeiten){
                this.tag = applikationszeiten.get("tag").getAsString();
                JsonParser mParser = new JsonParser();
                //JsonElement obj = mParser.parse(applikationszeiten.get("zeiten").getAsString());
                JsonArray mZeiten = applikationszeiten.get("zeiten").getAsJsonArray();
                this.zeiten = new ArrayList<String>();
                for (int i = 0; i < mZeiten.size(); i++) {

                        JsonObject zeit = mZeiten.get(i).getAsJsonObject();
                        this.zeiten.add(zeit.get("zeit").getAsString());
                }
        }
}
```

Aus dem Code ist zu entnehmen, dass die Modelle Beziehungen zueinander aufweisen. So ist die Applikationszeit als Attribut in der Verordnung vorhanden und die Verordnung als Attribut in dem Medikationsplan.

Sortieren und Filtern der Verordnungen

Pseudocode für das **Filtern** der Verordnungen für den aktuellen Wochentag:

```
public void filterVerordnungen(verordnungen){
	List result;
	for(int i=0;i<verordnungen.size();i++){
		if(verordnung[i].wochentag==aktuellerWochentag||verordnung.wochentag=="taeglich")
	result.push(verordnung[i]);

	}
	verordnungen = result;
}
```

Pseudocode für das **Sortieren** der Verordnungen nach den Zeiten (Insertionsort):

```
public void sortVerordnungen(verordnungen){
	time temp;
		for (int i = 1; i < verordnungen.length; i++) {
			temp = verordnungen[i].time;
			int j = i;
			while (j > 0 && verordnungen[j - 1].time > temp) {
				verordnungen[j] = verordnungen[j - 1];
				j--;
			}
			verordnungen[j] = temp;
		}
		return verordnungen;
}
```

Es ist zu beachten, dass die Zeiten richtig in Java-Objekte umgewandelt werden müssen.

Bencharichtigungen und Abonnieren von Topics

Das folgende Schema stellt den Verlauf der Nachrichten dar, die über RabbitMQ verteilt werden. Der Arzt-Client sendet in diesem Beispiel zwei Änderungen an Verordnungen zu zwei verschiedenen Patienten. Die Daten werden im JSON-Format vom Arzt-Client an den NodeJS-Server übermittelt. Der Producer, der NodeJS-Server, nimmt die Daten entgegen und sendet eine Nachricht mit den in der JSON mitgelieferten jeweiligen “station_id”, “patient_id” und der “beschreibung”. Der Routing-Key setzt sich dabei aus der station_id und der patient_id zusammen (station_id.patient_id). In diesem Fall sind die Routing-Keys “1.12” und “1.1”. Die Nachrichten werden auf den Exchange “amq.topic” gepublisht. Dieser leitet die Nachricht nun weiter auf alle Queues, die ein dem Routing-Key entsprechendes Binding besitzen.

In diesem Beispiel gibt es zwei Queues. Die Queue “personal1” hat ein Binding auf den Routing-Key “1,*”. Die Ziffer 1 steht für die Stations-ID. Das darauffolgende “*” - Symbol weißt darauf hin, dass die Queue alle Patienten der Station 1 abonniert hat. Die zweite Queue “patient1” besitzt ein Binding “1.1”. Die bedeutet, dass die Queue nur auf Nachrichten auf den Patienten mit der ID 1 auf der Station mit der ID 1 wartet.

Die Einfärbungen der Pfeile sollen die Verteilung der Nachricht hervorheben und wer die Nachricht erhält. Da das “personal1” alle Patienten der Station 1 abonniert hat, werden für diesen Consumer beide Nachrichten übermittelt.

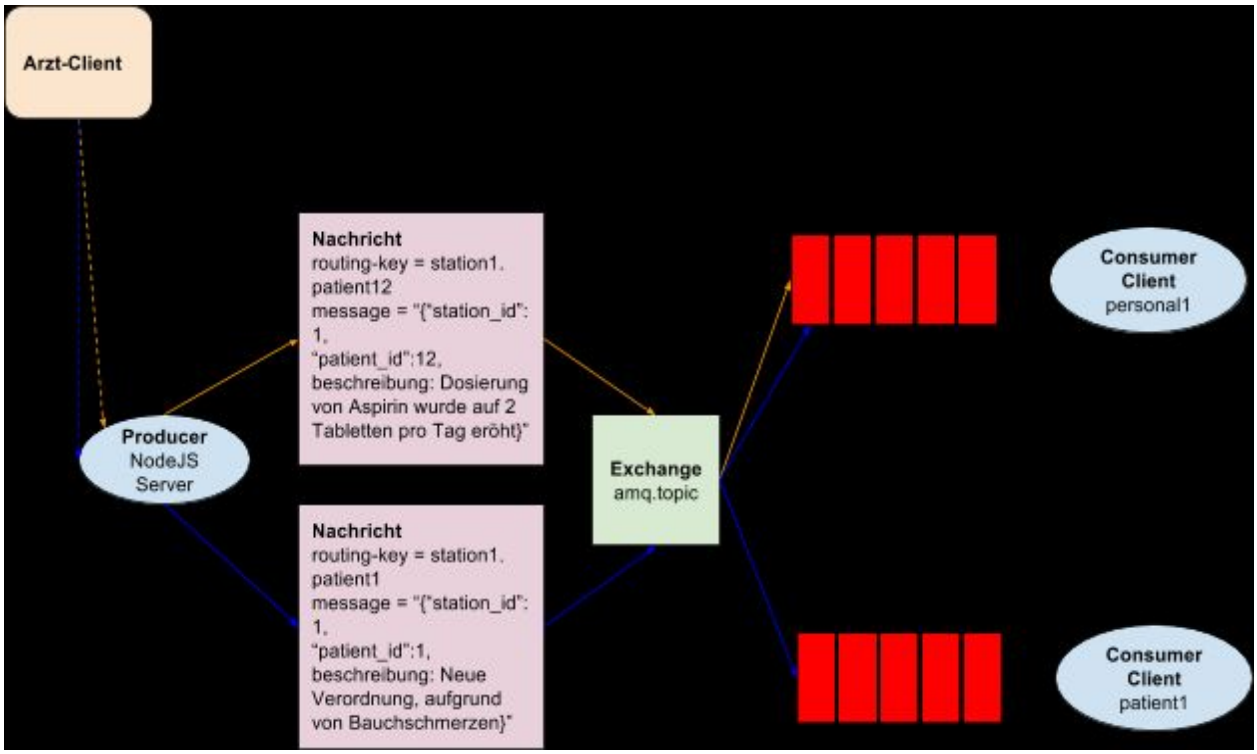

Abbildung 2

Medikationskontrolle

Das Medikamentenkontrollsystem wird über einen NodeJS Webservice simuliert. Der Webservice soll als überwachende Instanz fungieren, wenn Verordnungen vom Arzt angelegt werden. Dabei sollen die Daten des Patienten mit dem zu verordnenden Medikament verglichen werden, um unadäquate Verordnungen zu vermeiden und Wechselwirkungen festzustellen.

Die Methoden für die Medikationskontrolle werden auf dem NodeJS-Server programmiert. Bei dem Aufruf der Methoden werden Daten von dem Webservice bezogen.

Im folgenden wird Pseudocode verwendet, um die Anwendungslogik der Medikationskontrolle zu konkretisieren:

```
//**** Webservice **** //

// Liefert ein JSON mit allen Allergien zurück, die unverträglich mit dem Medikament sind
function getAllergieFromWebservice(medikament){
    var allergieList;
    webserviceconnection = mysqlconnect();
    webserviceconnection.query('SELECT allergie from Medikament WHERE medikament=?',[medikament],
function (err, rows) {
        if (!err) {
            allergieList = rows;
        }
        else {
            console.log('Error while performing Query.');
        }
    });
    webserviceconnection.end();

    return allergieList;
}

// Liefert ein JSON mit Nahrungsmitteln zurück, die unverträglich mit der Verordnung ist
function getNahrungsmittelFromWebservice(medikament) {
   var nahrungsmittelList;
    webserviceconnection = mysqlconnect();
    webserviceconnection.query('SELECT nahrung from Medikament WHERE medikament=?',[medikament],
function (err, rows) {
        if (!err) {
            allergieList = rows;
        }
        else {
            console.log('Error while performing Query.');
        }
    });
    webserviceconnection.end();

    return nahrungsmittelList;
}
```

```
// Überprüft die Wirkstoffe zweier Medikamente auf Wechselwirkung
function checkInteractions(medikament1, medikament2) {
    var medikament1Data;
    var medikament2Data;
    var result = {
        "medikament1":medikament1,
        "medikament2":medikament2,
    };

    webserviceconnection = mysqlconnect();
    webserviceconnection.query('SELECT * from Medikament WHERE medikament=?',[medikament1], function
(err, rows) {
        if (!err) {
            medikament1Data = rows;
        }
        else {
            console.log('Error while performing Query.');
        }
    });
    webserviceconnection.query('SELECT * from Medikament WHERE medikament=?',[medikament2], function
(err, rows) {
        if (!err) {
            medikament1Data = rows;
        }
        else {
            console.log('Error while performing Query.');
        }
    });
    webserviceconnection.end();

    //Liste der Wirkstoffe, die nicht mit diesem Medikament zusammen eingenommen werden dürfen
    interactionMedikament1Liste = JSON.parse(medikament1Data.interaction);

    for(var i=0;i<interactionMedikament1Liste.length;i++){
        if(interactionMedikament1Liste[i].name == medikament2Data.wirkstoff){
            // Medikament1 inkompatibel mit Medikament2
            result.push({"inkompatibel":medikament2Data.wirkstoff});
        }
    }

    // Überprüfen, ob die Medikamente die selben Wirkstoffe haben
    if(medikament1Data.wirkstoff==medikament2Data.wirkstoff){
```

```
        result.push({"risiko":"Gleicher Wirkstoff: "+medikament2Data.wirkstoff});
    }

    return result;

}

// Überprüft, ob das Medikament für schwangere Frauen geeignet ist
function checkPregnancy(medikament) {
    var pregnancyKompatibel;
    var result = {};
    webserviceconnection = mysqlconnect();
    webserviceconnection.query('SELECT pregnancy from Medikament WHERE medikament=?',[medikament],
function (err, rows) {
        if (!err) {
            pregnancyKompatibel = rows.pregnancy;
        }
        else {
            console.log('Error while performing Query.');
        }
    });
    webserviceconnection.end();

    if(pregnancyKompatibel!=1){
        // Ist nicht mit Schwangerschaften kompatibel
        result.push({
            "medikament":medikament,
            "kompatibel":false,
            "info":"Das Medikament"+ medikament+" darf nicht bei Schwangerschaften verabreicht werden."
        });
    }

    return result;
}

//******************************//
//**** Medikationskontrolle **** //
//******************************//

// Gibt ein JSON mit allen allergien des Patienten zurück
function getAllergie(patientID){
    var allergieList;
```

```
    mysqlconnection = mysqlconnect();
    mysqlconnection.query('SELECT name from Allergie WHERE patient_id=?',[patientID], function (err, rows) {
        if (!err) {
            allergieList = rows;
        }
        else {
            console.log('Error while performing Query.');
        }
    });
    mysqlconnection.end();

    return allergieList;
}

function getMedikamente(patientID){
    var medikamenteList;
    mysqlconnection = mysqlconnect();
    mysqlconnection.query('SELECT name from Verordnung NATURAL JOIN Medikament WHERE
patient_id=?',[patientID], function (err, rows) {
        if (!err) {
            allergieList = rows;
        }
        else {
            console.log('Error while performing Query.');
        }
    });
    mysqlconnection.end();

    return medikamenteList;
}

// Überprüft, ob die Allergien verträglich mit einem Medikament ist
// Liefert ein JSON mit Liste der unverträglichen Allergien zurück
function checkAllergie(medikament, patientID) {
    // Liste der Allergien, die mit dem Medikament nicht verträglich sind
    var allergieListFromMedikament = getAllergieFromWebservice(medikament);
    // Liste aller Allergien, die unter denen der Patient leidet
    var allergieListFromPatient = getAllergie(patientID);
    // Liste der Allergien, die sich mit nicht dem Medikament vertragen
    var gefundeneAllergien = {};
    // Suche nach gemeinsamen Allergien der Listen
    for (var i = 0; i < allergieListFromMedikament.length; i++) {
```

```
        for (var j = 0; j < allergieListFromPatient.length; j++) {
            if (allergieListFromMedikament[i] == allergieListFromPatient[j]) {
                // Füge Allergie zur Liste hinzu
                gefundeneAllergien.push(allergieListFromPatient[j]);
            }
        }
    }
    return gefundeneAllergien;
}

// Liefert eine vollständige Liste aller Nahrungsmittel zurück, die nicht unverträglich mit den Verordnungen des Patienten sind
function getNahrungsmittelListe(patientID){
    var medikamente = getMedikamente(patientID);
    var nahrungsmittelListe ={};
    for(var i = 0; i < medikamente.length; i++) {
        nahrungsmittelListe.push(getNahrungsmittelFromWebservice(medikamente[0].name));
    }
    return nahrungsmittelListe;
}
```