Medical Assistent

Adrian Locher, Jason Benz

26. Oktober 2021

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die Frage, ob ein Arzt besucht werden soll, ist häufig ein Dilemma. Einerseits will man nicht diejenige Person sein, welche bei unbedenklichen Symptomen überreagiert, andererseits könnten sich die Symptome verschlimmern und vielleicht hätte der Arztbesuch das Problem frühzeitig beheben können. Dazu kommt die Krankenversicherung, welche einen Teil der anfallenden Kosten übernimmt und eher ein Argument für ein Arztbesuch ist. Aber aus Sicht der Krankenkasse verursacht jede überflüssige Konsultation des Arztes unnötige Kosten für die Allgemeinheit.

Eine ganz andere Beudeutung hat dieses Dilemma ausserdem seit dem Anfang der Covid-19 Situation. Ärzte und anderes Gesundheitspersonal, gelten als besonders stark ausgelastet. Auf der Kehrseite hingegen, sollte man bei Covid-19-artigen Symptomen auch nicht zögern und sich testen lassen. Welche Symptome nun genau als „Covid-Symptome“ gelten und welche nicht, ist teilweise sehr unübersichtlich, was die Lage auch nicht einfacher werden lässt.

Das primäre Ziel unseres Projekts ist eine Lösung zu entwickeln, mit der unnötige Arztbesuche generell, aber besonders in Situationen wie den aktuellen, vermieden werden können. Es ist nicht beabsichtig die allgemeine Zahl der Arztbesuche zu verringern, sondern in Grenzfällen bei der Entscheidung zu helfen.

Um dabei auch wirklich das Gesundheitspersonal maximal zu entlasten, besteht unser Lösungsansatz nicht aus einer Hotline oder ähnlichem. Ziel ist es ein AI gestütztes System zu entwickeln, welches ganz ohne Menschliche Interaktion auskommt. Mithilfe von Googles Dialogflow Plattform, sowie einem interaktiven Java Client, wurde ein Chatbot realisiert, der diese Kommunikation übernimmt.

2 Backend

2.1 Übersicht

In Abbildung 1 ist der logische Ablauf, des Dialogflows zu sehen, welcher das Backend unseres Chatbots bildet.

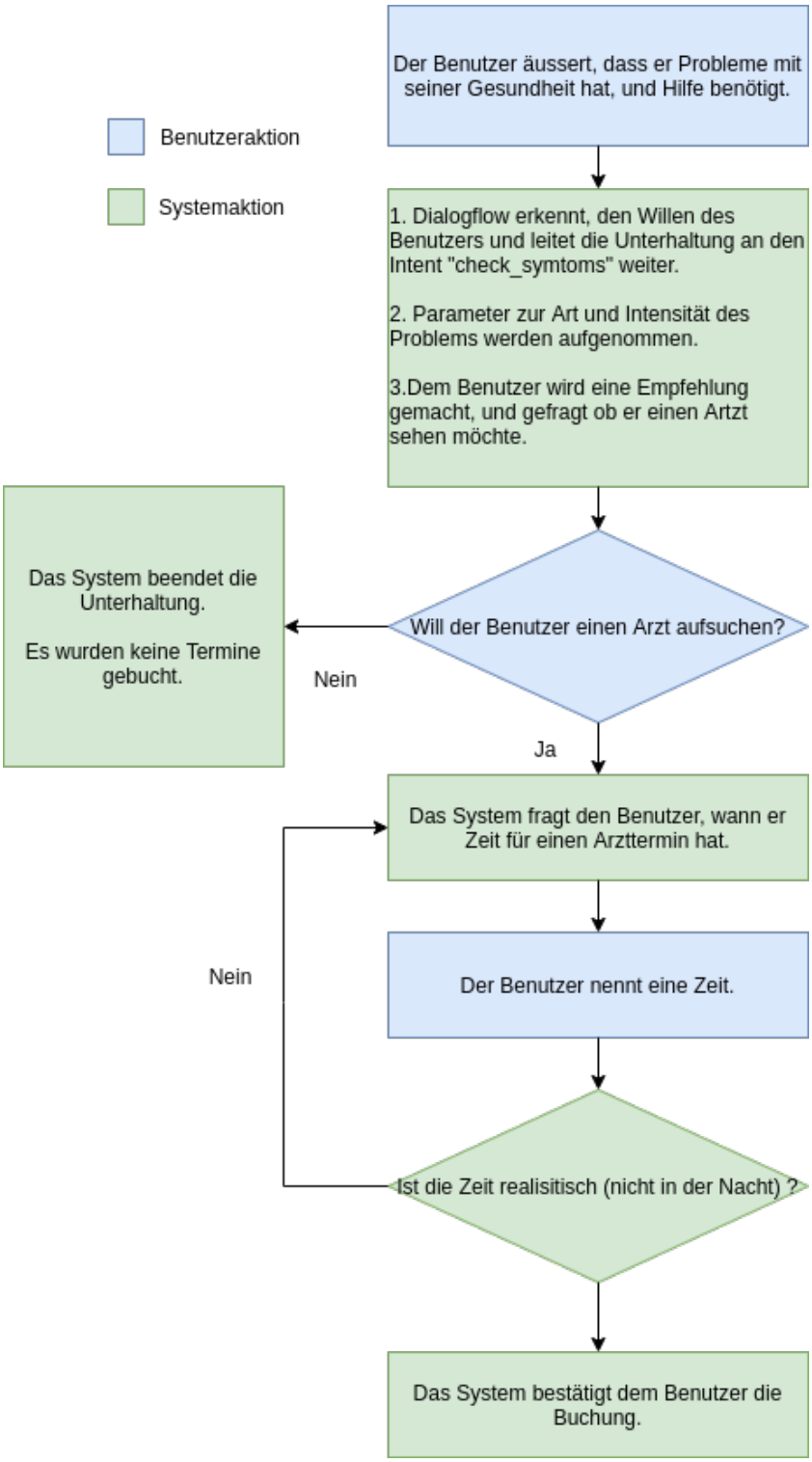

Abbildung 1: Backend Übersicht-Flowchart

2.2 Intents

Das Backend ist in vier Intents strukturiert:

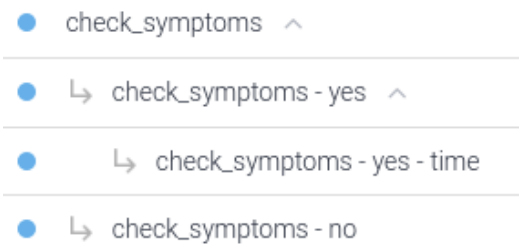

Abbildung 2: Dialogflow Intent-Liste

2.3 Aufnahme von Symptomen

Im ersten Intent des Dialogs *„check_symtpoms“* wird eine Liste mit Symptomen, sowie ein Schmerzniveau und die Zeitdauer seit dem ersten Erscheinen der Symptome aufgenommen.

Folgende Trainingssätze wurden für das Auslösen des Intents definiert:

- i have *pain* since *yesterday*
- i have *pain*
- i need help
- should i go to the doctor?
- Can you tell me if i should seee a doctor?
- Am I sick?
- I feel Sick
- feel *unwell*

Die hier *kursiv* geschrieben Wörter werden auf die entsprechenden Parameter gematched.

2.3.1 Parameter Entities

Um die Symptom-Parameter aufzuzeichnen wurden folgende Parameter und Entities definiert:

Parameter	**Entity**
SymType	symptom_type
SymIntensity	symtom_intensity
SymDuration	System.Date

Tabelle 1: Paramter und Entities

symtom_type hat einen Wertebereich von: pain, unwell, sick, fever, cough, loss of taste, chestpain, tiredness und shortness of breath mit jeweils drei bis vier Synonymen.

symtom_intensity hat einen Wertebereich von 1 bis 5. Diese können jeweils in Worten (*z. B. „One“, „Eins“ usw.*) oder als Zahl (*z. B. 1*) angegeben werden.

System.Date ist ein vorgefertigtes Entity mit einem Wertebereich, welcher beliebige Kalenderdaten enthalten kann.

2.4 Auswertung der Symptome

In einem zweiten Schritt geht es nun um die Auswertung der Angaben, damit dem Patienten Tipps zum weiterem Verhalten gegeben werden können. Die Auswertung findet am Ende des Intents statt, indem ein Webhook aufgerufen wird. Hier ist als Antwort auf dem entsprechenden Intent *„check_symptoms“* eine Funktion *„checkSymptoms“* (siehe: Abbildung 3) registriert.

```
1  const coronaSymptoms = ['cough', 'loss of taste', 'fever', 'tiredness', 'shortness of breath', 'chestpain'];
2  function checkSymptoms(agent) {
3    let coronaSymptomsCount = 0;
4    for(let sym of agent.parameters.SymType) {
5        if(coronaSymptoms.includes(sym)) {
6            coronaSymptomsCount++;
7        }
8    }
9
10   if(coronaSymptomsCount > 3) {
11       response.json({
12         payload: { hasMaybeCorona: true },
13         fulfillmentText: "It seems, you might have caught the corona-virus."
14                        +  "You should isolate imediately, and see a doctor as"
15                        +  " soon as possible. Would you like to book an appointment now?"
16       });
17   } else {
18       if(Number(agent.parameters.SymIntensity) <=2) {
19           agent.add("Your symptoms, don't seem to be very bad yet, we suggest"
20                   + " you wait a little and contact us, if they get worse."
21                   + " Would you still like to see a doctor?");
22       } else if(Number(agent.parameters.SymIntensity) == 3) {
23           agent.add("Your concern seems to be justified. Would you like to see a doctor?");
24       } else {
25           agent.add("We strongly suggest you, to see a doctor. Would you like to book an appointment now?");
26       }
27   }
28
29 }
```

Abbildung 3: checkSymptoms Funktion

Auf den Zeilen 3 bis 8 werden die angegebenen Symptome mit dem Array *coronaSymptoms* verglichen und gezählt. Auf Zeile 10 wird überprüft, ob die Anzahl der auf Corona-Symptome passenden Symptome grösser als 3 ist. Falls dies der Fall ist - wird unabhängig der anderen Parametern - eine entsprechende Nachricht zurückgesandt, welche den Benutzer über diesen Umstand informiert und ihm vorschlägt, sich zu isolieren, sowie sich testen zu lassen.

Besteht kein Verdacht auf das Corona-Virus, so ist die Symptomstärke (*„symtom_intensity“*) ausschlaggebend für die Antwort:

- **Intensität <= 2:** Die Symptome scheinen nicht schlimm zu sein. Dem Benutzer wird geraten abzuwarten und bei Verschlimmerung des Zustands einen Arzt aufzusuchen. Der Benutzer wird gefragt, ob er dennoch einen Arzttermin buchen möchte
- **Intensität = 3:** Der Benutzer wird gefragt, ob er einen Arzttermin buchen möchte.
- **Intensität > 3:** Dem Benutzer wird angeraten einen Arzt aufzusuchen. Er wird gefragt, ob er einen entsprechenden Termin buchen möchte.

Alle Antworten resultieren in der Frage, ob der Benutzer einen Arzttermin buchen möchte. Mit den Intents *check_symptoms - yes* und *check_symptoms - no* (siehe: Abbildung 2) wird die Antwort des Benutzers abgefangen. Während *check_symptoms - no* zu einer Verabschiedungsantwort führt, fragt *check_symptoms - yes* direkt, wann der Termin stattfinden soll.

2.5 Termin buchen

Der Intent *check_symptoms - yes - time* fängt die Antwort des vorgehenden Intents (Frage nach Terminzeit) direkt ab und löst im Webhook eine weitere Methode aus (siehe: Abbildung 4).

```
1  function handleAppointment(agent) {
2      let date;
3      if(agent.parameters.AppointmentTime.date_time) {
4        date = new Date(agent.parameters.AppointmentTime.date_time);
5      } else {
6          date = new Date(agent.parameters.AppointmentTime);
7      }
8      if(date.getHours() >= 18 || date.getHours() < 8) {
9          agent.add("I'm sorry, but there are no appointments at night,"
10                 + " please specify another Appointment.");
11         agent.followup_event={'name':'set_appointment_time'};
12     } else {
13         agent.add(`You have booked an appointment at: ${date.toString()}`);
14     }
15 }
```

Abbildung 4: handleAppointment Funktion

Die ersten 7 Zeilen der Funktion kümmern sich um das Parsen des Datums. Da dieses nicht immer gleich daher kommt, musste ein wenig Logik zur Überprüfung der Daten implementiert werden. Zeile 8 prüft, ob die Uhrzeit zwischen 08:00 und 20:00 Uhr liegt. Alles zwischen diesen zwei Zeitpunkten ist als gültige Uhrzeit definiert und führt dazu, dass die Buchung auf Zeile 13 bestätigt wird. Der Dialog ist hiermit beendet. Liegt die Zeit allerdings ausserhalb des gültigen Bereichs, wird auf den Zeilen 9 und 10 eine Fehlernachricht zurückgegeben und nach einer neuen Zeit gefragt. Auf Zeile 11 wird als Folge-Intent das Event *set_appointment_time* definiert. Dieses Event zeigt auf denselben Intent, auf dem wir gerade operieren (*check_symptoms - yes - time*). Die Terminbuchung wird also wiederholt, bis eine gültige Zeit für den Termin angegeben wird.

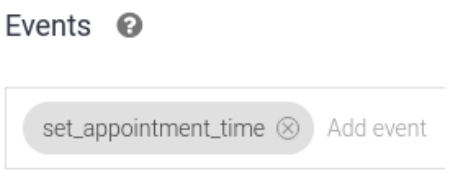

Abbildung 5: Eventeinstellung für die Schleifenlogik

3 Java Client

Der Java Client ist in die folgenden drei Bereiche aufgeteilt, welche in den nachfolgenden Unterkapitel genauer beschrieben werden:

1. Command Line Interface
2. Dialogflow Connector
3. Medical Assistant

3.1 Command Line Interface

Beim Start des Medical Assistants im Java Client wird der User auf folgende Möglichkeiten hingewiesen:

- Input mit dem medizinischen Problem angeben
- „q“ um den Assistenten wieder zu beenden
- „v“ um den persönlichen medizinischen Bericht anzuschauen

Die logische Abfolge des Benutzerinputs, bzw. die Interaktion mit dem Dialogflow Chatbot ist analog zum beschriebenen Ablauf in Abildung 1. Nach jeder Antwort des Chatbots können die Optionen *„q“* (beenden) und *„v“* (Report anschauen) aufgerufen werden.

3.2 Dialogflow Connector

Beim Projekttyp handelt es sich um ein Maven-Projekt. Für die Interaktion mit Dialogflow wurde das Artefakt *„google-cloud-dialogflow“* verwendet.

Die Einbindung, sowie die Basisimplementierung (Dialogflow Kommunikation) war bereits Bestandteil einer Übungsstunde. Deshalb wird dies als Vorwissen angeschaut und in diesem Report nicht weiter thematisiert. Projektspezifische Anwendungsfälle werden in den entsprechenden nachfolgenden Kapiteln beschrieben.

3.3 Medical Assistant Implementation

3.3.1 Report

Neben der Interaktion mit dem Dialogflow Chatbot ist das Hauptziel des Java Clients, dem Benutzer einen medizinischen Bericht zur Verfügung zu stellen. Wenn dieser mit dem CLI Kommando *„v“* abgerufen wird, ist dieser zu Beginn leer:

```
Hello, I'm your medical assistant. How can I help you?
Please enter your request and confirm with enter. Other options: q to quit, v to view your medical report

v
Symptom type(s):    -
Is maybe corona:    -
Symptom intensity:  -
Symptom start:      -
Symptom duration:   -
Doctor appointment: -
```

Abbildung 6: Leerer medizinischer Report

Fortlaufend - basierend auf den User-Inputs, sowie den Chatbot Antworten - wird der Report ergänzt (siehe: 3.3.2). Am Ende kann der ausgefüllte Bericht zum Beispiel folgendermassen aussehen:

```
Symptom type(s):    [fever, cough, shortness of breath, tiredness]
Is maybe corona:    true
Symptom intensity:  5
Symptom start:      18.10.2021
Symptom duration:   7
Doctor appointment: 26.10.2021 - 12:00
```

Abbildung 7: Komplett ausgefüllter Beispiel-Report

3.3.2 Intent Handling

Die für den Java Client relevanten Intents sind *„check_symptoms“*, sowie der Folge-Intent *„check_symptoms - yes - time“*. Via Switch-Case Statement werden Intent-spezifische Aktionen durgeführt, welche schlussendlich den medizinischen Bericht aktualisieren.

Intent „check_symptoms“

1. Via Dialogflow Backend wird dynamisch anhand der Symptome ermittelt, ob der Benutzer eventuell Corona hat. Diese Information wird über den Payload an den Java Client übermittelt. Das Query-Result im Java Code enthält die Methode *„getWebhookPayload()“*, welche den Zugriff auf den Payload als Struct ermöglicht. Auf dieser Datenbasis wird der Report entsprechend um das Attribut *„hasMaybeCorona“* ergänzt.

```
public void checkCorona(Struct webhookPayload) {
    var payload : Set<Map<K, V>.Entry<String, Value>> = webhookPayload.getFieldsMap().entrySet();
    for (Map.Entry<String, Value> entry : payload) {
        if (entry.getKey().equals("hasMaybeCorona")) {
            this.hasMaybeCorona = entry.getValue().getBoolValue();
        }
    }
}
```

Abbildung 8: Corona Payload zum Report hinzufügen

2. Im nächsten Schritt werden alle aktualisierten Parameter auf dem Report im Java Client hinterlegt. Nachfolgend ist ein Ausschnitt der Aktualisierung des ersten Parameters zu sehen. Alle anderen Parameter werden im Swicht-Case Statement analog zu diesem Beispiel aktualisiert.

```
public void updateHealthInfo(Struct parameters) throws ParseException {
    Set<Map.Entry<String, Value>> entries = parameters.getFieldsMap().entrySet();

    for (Map.Entry<String, Value> entry : entries) {
        Value value = entry.getValue();

        switch (entry.getKey()) {
            case "SymIntensity":
                if (!value.getStringValue().isEmpty()) {
                    setSymptomIntensity(Integer.parseInt(value.getStringValue()));
                }
                break;
```

Abbildung 9: Ausschnitt der Report Aktualisierung

Intent „check_symptoms - yes - time“

1. In diesem Intent wird noch der Arzttermin auf dem Report aktualisiert.

```
public void updateAppointment(Struct parameters) throws ParseException {
    var appointment : String = parameters.getFieldsMap().get("AppointmentTime").getStringValue();
    doctorAppointment = new SimpleDateFormat( pattern: "yyyy-MM-dd'T'HH:mm:ssXXX").parse(appointment);
}
```

Abbildung 10: Arzttermin auf dem Report hinzufügen

4 Discussion

4.1 Was wir gelernt haben

Bei der Umsetzung dieses Projekts haben wir gelernt, dass die Dialogflow Plattform sich äusserst gut eignet, um Schnittstellen zwischen Menschen und Informatiksystemen zu konstruieren. Die so erschaffenen Benutzeroberflächen sind - wenn richtig umgesetzt - sehr intuitiv zu bedienen. Sehr viel Potential bietet diese Technologie aus unserer Sicht auch, um Menschen mit körperlichen Einschränkungen den Zugang zu diversen Programmen zu verschaffen. Das bekannteste Beispiel sind dabei moderne Sprachassistenten.

4.2 Limitationen

Wenn man an AI denkt, geht man schnell davon aus, dass sich eine Interaktion mit einem System kaum noch von einer Interaktion mit einem Menschen unterscheidet. Dies mag vielleicht in Zukunft der Fall sein, ist es aber heute noch nicht.

Dialogflow, kommt schnell an seine Grenzen, wenn man den Chat-Bot mit ungenauen oder zusätzlichen (unnötigen) Informationen beliefert, für die er nicht trainiert wurde.

4.3 Erweiterungsmöglichkeiten unserer Arbeit

Um die Aufnahme der Informationen weiter zu verbessern, könnte der Satz an Trainingssätzen erweitert werden, sodass es weniger häufig zu Missverständnissen durch einfache Schreibfehler kommt.

Weiter könnte man unseren Chatbot durch folgende Features erweitern, um ihn für einen produktiven Einsatz vorzubereiten:

- Buchungssystem mit persistenten Daten und Konfliktauflösung, um doppelte Terminbuchungen zu verhindern.
- Automatische Zuweisung von Patienten an Ärzte mit unterschiedlichen Spezialisierungen, basierend auf den angeggebenen Symptomen.
- Implementation eines Webinterfaces oder Interface mit einem Sprachassistenten.
- Integration in Chat-App, um eine vereinfachte Kommunikation zu ermöglichen (z. B. Facebook Messenger oder Telegram)