README.md

Technical Description - Hazel

Hazel repository on Github

Abstract

The idea behind Hazel is to become a intelligent centralized health care information service. We want it to be able to answer any health care related question with either an educated answer or direction to the right service the patient should use.
For the prototype we decided to implement an AI personal assistant which will be integrated with one of the available personal assistants and a website for hosting rich content pages where health care services are showcased.

Prerequisites

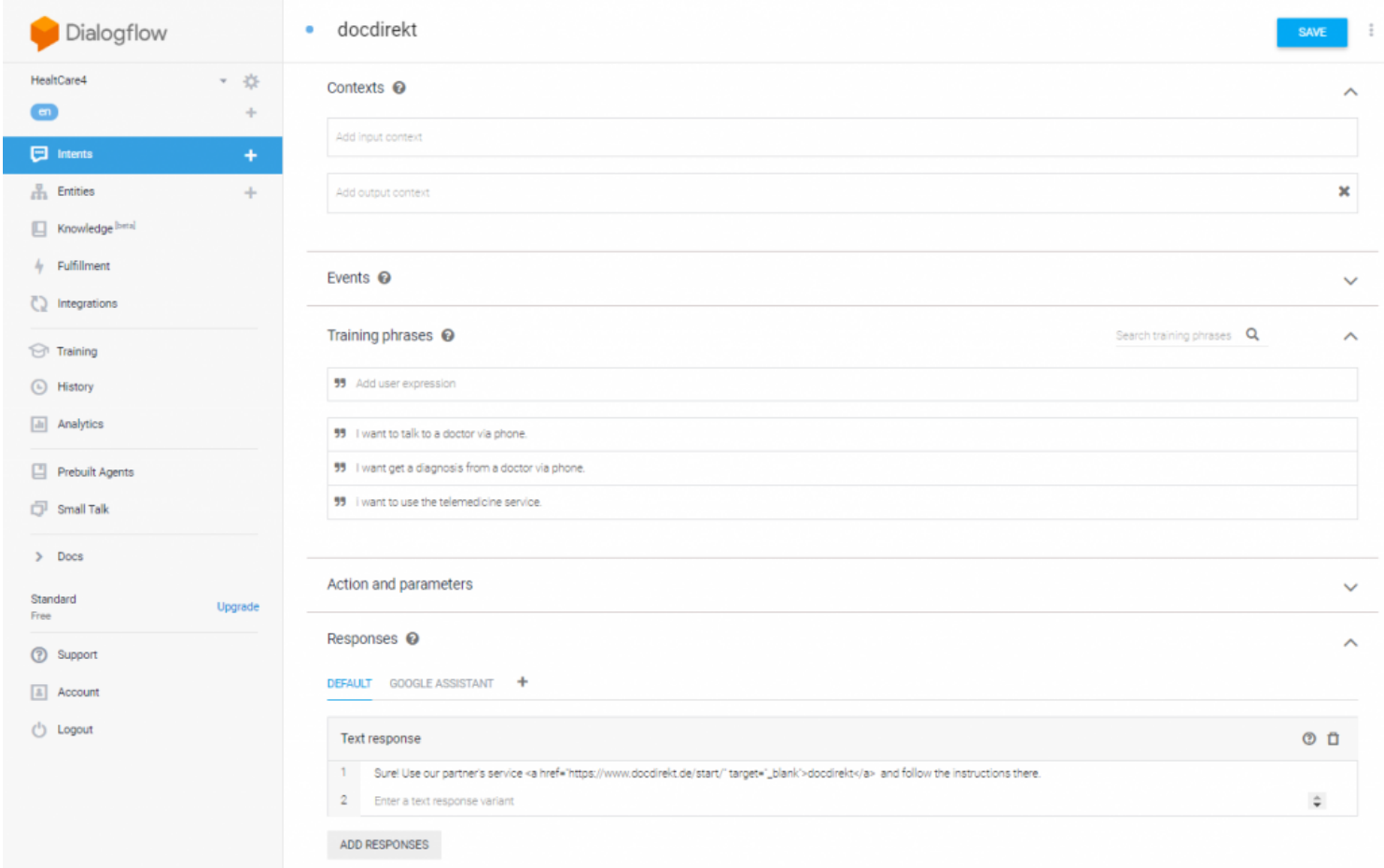
- AWS Account
  - EC2 Machine with enough computing power to hold the server
  - Static IP
- Google developer account with the following services available
  - Google Dialogflow
  - Google Actions
- Freenom account with domain reserved
- WordPress Account

Personal Assistant

The personal assistant is supported by a tool provided by Google for Natural Language Understanding (NLU) called "Dialogflow". The most current non-beta version we use is V2. Dialogflow takes over the function to process natural language input for the personal assistant. Then answers are selected or generated that are either static or have minimal logic. In the following the term "Dialogflow Agent" or "Agent" is used. In this context it is a term for a project in Dialogflow. This is where the intents of the chatbot are stored. An Intent is the framework, in which the information for the question treatment is stored. A Dialogflow Agent typically contains several intents to be able to fulfill all tasks assigned to it. An intent is called by the user via a suitable input. The agent then delivers a corresponding answer. The information received from the user can also be further processed. The following information can be stored in an intent in addition to its name (the sequence corresponds to that in Dialogflow):

- Contexts
- Events
- Training Phrases
- Action and parameters
- Responses
- Fulfillment

The following screenshot shows the Dialogflow console in which the Intent is created and worked on:



The training files and webhooks we used can be found on the Dialogflow folder.
The assistant was manually trained to support limited set of scenarios, links and interaction with user was splitted to HTML based responses which are used in the website integration, and to card and other special Google Assistant responses which are used in the Google Integration responses. Where more logic needed to be implemented in response we used webhooks which are hosted directly on our Dialogflow account.

In the following sections we will explain the individual components in further details.

Contexts

Contexts are very important for the flow of conversation. A distinction is made between "input context" and "output context". An "output context" activates the context entered there when the intent is called. It must be defined how long the context should be active before it automatically becomes inactive again. An "input context" ensures that the corresponding intent can only be called if the specified context is active. Using this function, the agent can remember which intents have already been accessed and then activate others. This enables a more personalized and thus more natural call process. If an "output context" for an intent is unique, an intent with the corresponding "input context" is called a "follow-up intent".

Events

Besides training phrases, events are another way to control an intent. Here, an intent can be triggered without direct action by the user. Dialogflow offers a selection of events.

Training Phrases

The training phrases serve the machine learning of the agent. Possible user entries are stored here for each intent if the user wants to access this intent. Dialogflow recommends 10-20 training phrases per intent in its guides. Based on these training phrases and with the help of Machine Learning, Dialogflow creates a dynamic model. The user's inputs are then assigned to the intents using this model.

Actions and parameters

In the line for "actions", values can be defined that are to be sent via the webhook. The section "parameters" is used to annotate the training phrases. Entities representing a group of terms can be defined. In this group the different terms with their synonyms can be defined. In the training phrases words or phrases can be marked and assigned to the entities. This annotation of the training phrases helps the dynamic model of the agent to better understand what is being said. In addition, parameters, i.e. the entity values contained in what is said, can be extracted and used further.

Responses

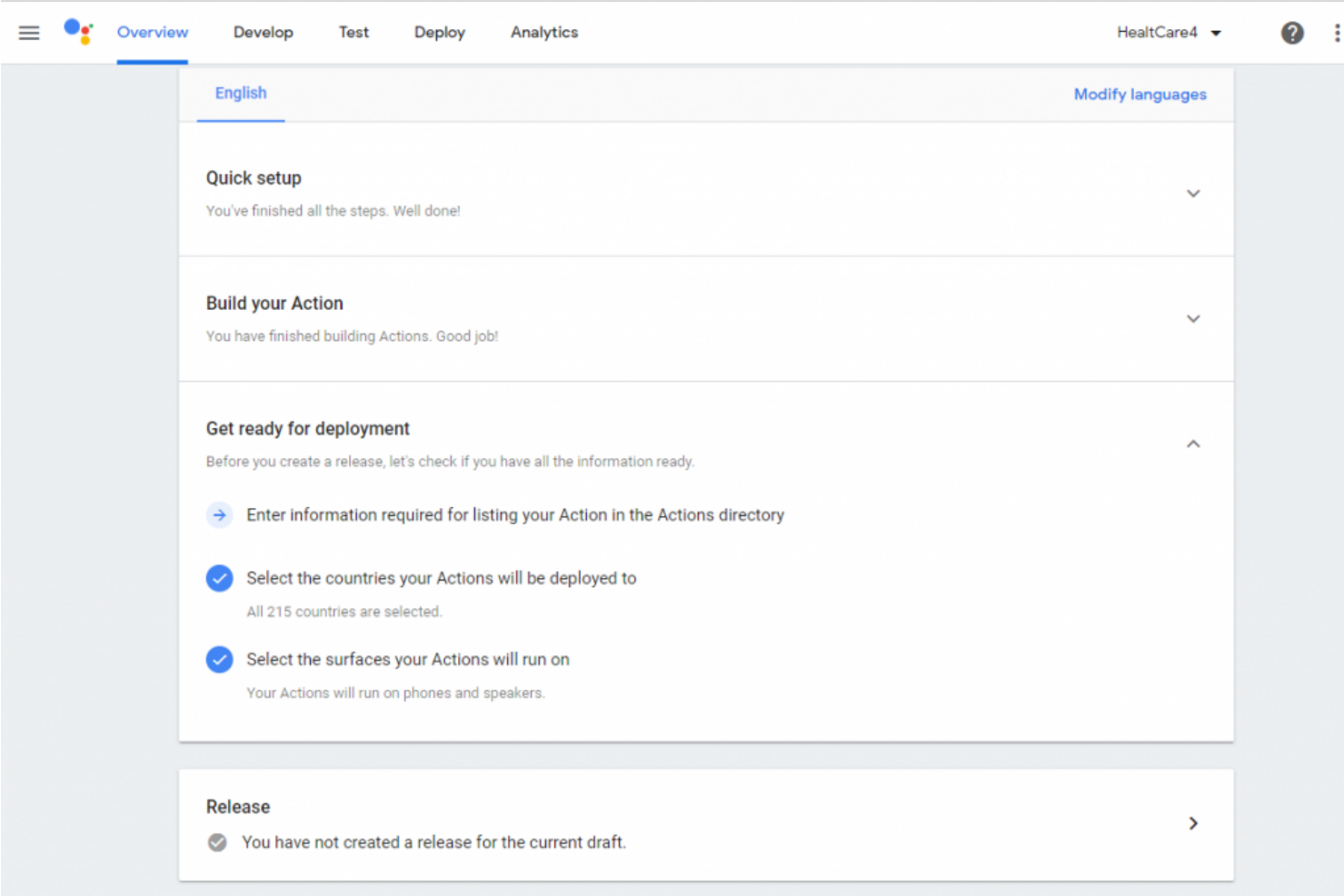
Here you define the response that Dialogflow should return. The possibilities here are manifold. Several variants of a text answer can be specified. Dialogflow then goes through the individual variants when answering. If no variant is left for a query, the process starts again from the beginning. This feature helps to make the agent look more human.

Fulfillment

A webhook can be used via fulfillment. A webhook is an HTTP callback. An application that implements a webhook will send an HTTP POST to the URL of the webhook when certain events occur. With a webhook you don't have to constantly ask if there is new information, but the data is transferred directly as soon as it is available. The webhook can also process the received data itself and trigger actions that are not related to the original event. Dialogflow supports the integration of a webhook. The use of a webhook is called "fulfillment" in Dialogflow. With the help of fulfillment, a Dialogflow agent can call the business logic for specific intents. Thus, during the course of the conversation, the information extracted by Dialogflow's natural speech processing can be used to generate dynamic responses or, as mentioned above, to trigger further actions in the backend. For example, the webhook can be used for answers that require current information from the database. In our case we used the Inline Editor (Powered by Cloud Functions for Firebase) provided by Dialogflow to implement the webhook.

Google Assistant Integration

For the prototype, we choose to integrate it with Google Assistant since it seems to be the most accessible and familiar.
The way to integrate application with Google Assistant is to use the Google Action Console, which let define an "Action" which is an extra capabilities set for the Google Assistant.



Using the console we have created an Action project from our Dialogflow model. The console is where all information and configuration needed for deployment is found. We have trained a trigger event from the Action console for 'Please Talk to Hazel Help' and we launch our test version of the Google Assistant so it can be used from any user we have attached to the test version. After attaching our usernames on the console we could test our application on our phones, tablets or smart-home device.

Website Integration

The website integration is done using 'My Chatbot' Wordpress extension, the extension lets us easily implement a view which connects to Dialogflow API using a shortcode within the Wordpress pages.

Website

The website is hosted on EC2 AWS instance with route 53 static IP (52.29.18.178) The machine is running Ubuntu 18.04

After launching the instance, Wordpress + Apache server need to be installed see manual. As part of the installation MySQL server would also be installed on the machine.

The website is using the following Wordpress plugins

- Elementor for a user friendly building of weg pages

- Insert Headers and Footers, and Scripts n Styles for adding custom javascripts
- My Chatbot to integrate our Dialogflow model in the website
- Participants Database for viewing databases tables on pages
- Really Simple SSL
- Super Progressive Web Apps and AMP to support PWA (website as an app)
- wpDiscuz for improving the commenting system and add rating
- TablePress for creating tables
- RumbleTalk Chat for adding chat rooms and enables users to chat with each other
- Elementor - Header, Footer & Blocks to add a customized header

The domain is served using Freenom, after buying static IP from AWS route 53 and attaching our server to it, on the Freenom console we have registered the static IP on 'A' record for our domain DNS.
For the SSL certificate we used Let's Encrypt, after getting the certificate we have also registered it on the server and on 'TXT' records on the DNS server.

Example pages was created for number of doctors which can be redirected from the personal assistant, we used Elementor template for easily creating multiple pages.

For showing the waiting-times there's an MySQL server running on our server which holds the following table:

> Name [Text-line], Hospital [Checkbox], Waiting Time [Numeric], Address [Text-line], City[Text-line], State [Text-line], Country [Text-line], Zip [CodeText-line], Phone [Text-line]

We have populated the table with example facilities from Munich.
We have created dedicated page which has the view of the facilities database, with option to view/hide hospitals, and to find specific facility by location (using shortcode of Participants Database extension and javascript). The waiting times are updated using a cron job on the server that is running a python script periodically which updates the waiting time - this should be the point of interface for getting the waiting times.

Scripts

We decided to use Python as our scripting environment on the server. the different scripts which were used to automate some of the tasks and to run on the servers are on the scripts folder.