# *SENSECHAT*

Internet of Things

*Robin Bekaert – 2NMD*

Docent: Frederick Roegiers
*2019 - 2020*

INHOUD

## 1. Discover

Voor deze opdracht heb ik gekozen om een chatbot te maken die gelinkt is aan de Sensehat van de Raspberry Pi. Ik heb hiervoor gekozen omdat het mij wel een leuke toepassing leek om de Sensehat te integreren in mijn werk. Wegens een heel aantal bugs is de verbinding op het moment van deze deadline echter nog niet gelukt.

Voor de uitwerking van de chatbot heb ik ervoor gekozen om de Python library “Chatterbot” niet te gebruiken. Het leek mij interessanter om zelf de bot te schrijven. Hierdoor heb ik wel minder mogelijke vragen en antwoorden, maar het is meer gepersonaliseerd. Het is een soort Q&A bot die vragen over mezelf kan beantwoorden, maar ook a.d.h.v. enkele API’s bijvoorbeeld de huidige temperatuur kan geven of een interessant kattenweetje kan vertellen. Er wordt machine learning gebruikt om een zo accuraat mogelijk resultaat te bekomen.

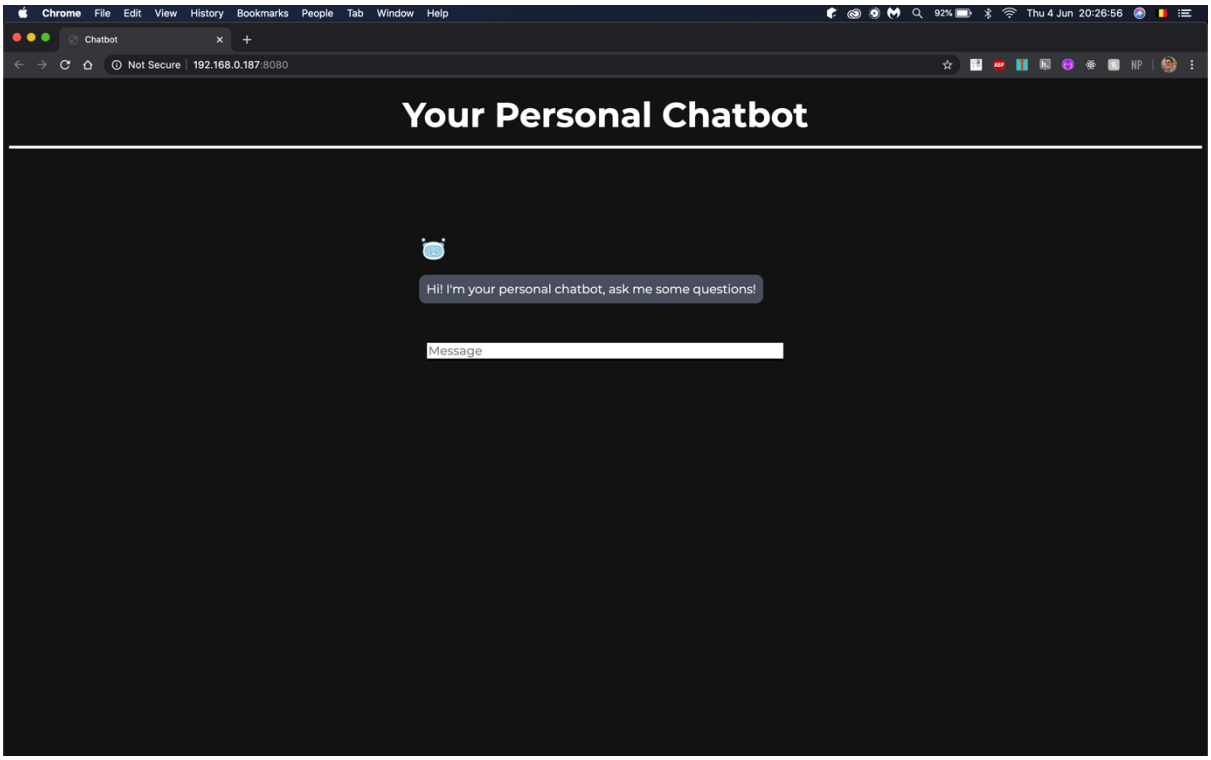
## 2. Define

Er is niet veel hardware nodig voor dit project, enkel een Raspberry Pi met een Sensehat. Om deze bot te schrijven heb ik gebruik gemaakt van Python. De lijst met onderwerpen waar de bot over kan praten is opgeslagen in een json-file.

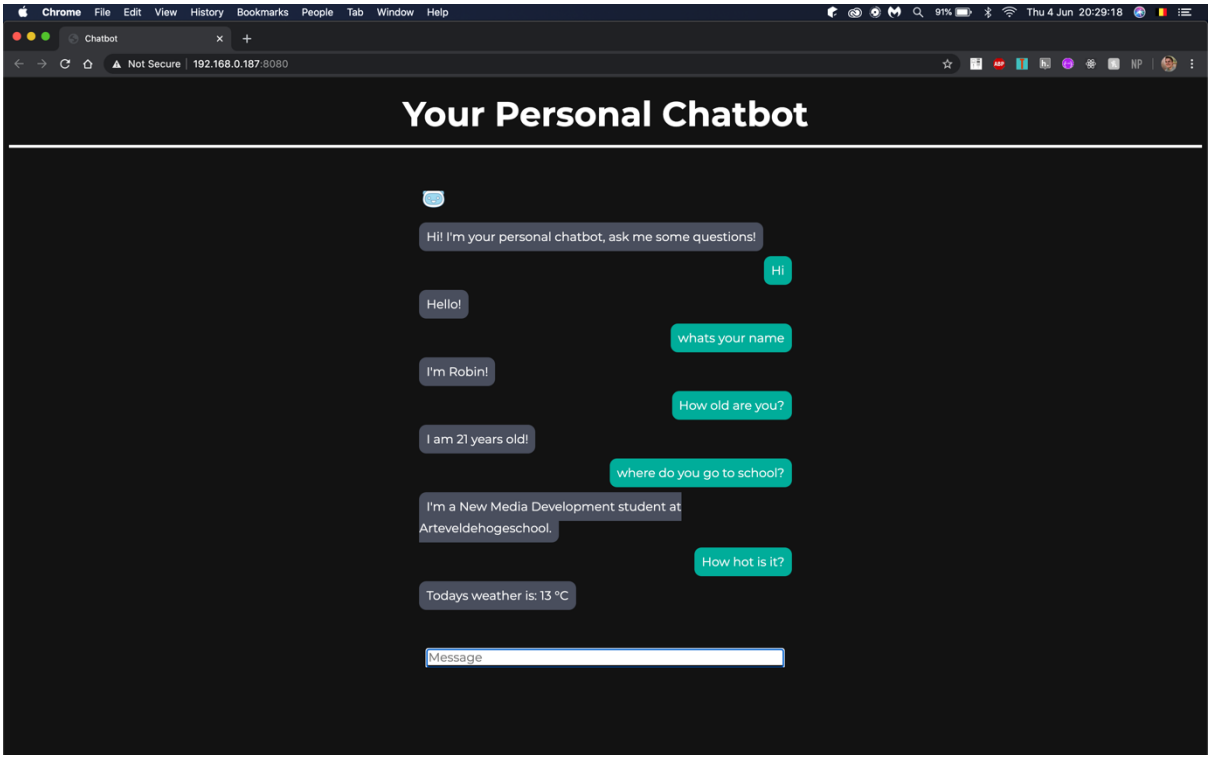
Ik heb gebruik gemaakt van een aantal Python modules om de bot te implementeren:

- Nltk om de input van de gebruiker te verwerken. Elke zin die de gebruiker typt wordt teruggebracht tot de stam via de Lancasterstemmer.
- Tensorflow en TFLearn voor het machine learning aspect van de chatbot. Het model wordt getraind met enkele mogelijke vragen en de bijbehorende antwoorden. Op basis hiervan wordt dan bij elke vraag van de gebruiker het meest accurate resultaat gekozen.
- Flask om een online dashboard te creëren om met de chatbot te communiceren.

## 3. Design



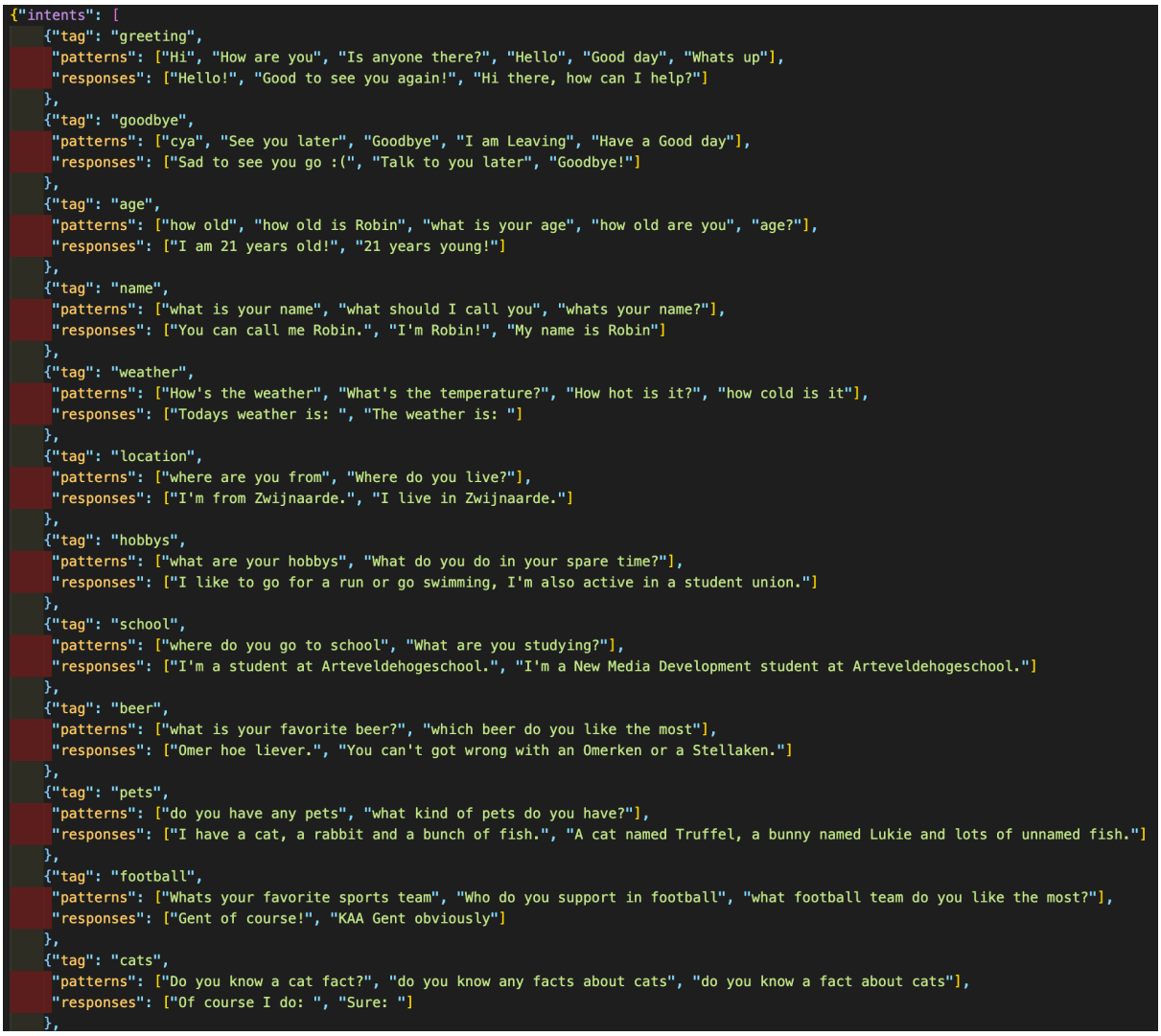

Dit is het scherm dat je te zien krijgt wanneer je de bot opstart.




Zo ziet een korte interactie me de bot eruit.

# 4. Develop

## intents.json



```
{"intents": [
    {"tag": "greeting",
     "patterns": ["Hi", "How are you", "Is anyone there?", "Hello", "Good day", "Whats up"],
     "responses": ["Hello!", "Good to see you again!", "Hi there, how can I help?"]
    },
    {"tag": "goodbye",
     "patterns": ["cya", "See you later", "Goodbye", "I am Leaving", "Have a Good day"],
     "responses": ["Sad to see you go :(", "Talk to you later", "Goodbye!"]
    },
    {"tag": "age",
     "patterns": ["how old", "how old is Robin", "what is your age", "how old are you", "age?"],
     "responses": ["I am 21 years old!", "21 years young!"]
    },
    {"tag": "name",
     "patterns": ["what is your name", "what should I call you", "whats your name?"],
     "responses": ["You can call me Robin.", "I'm Robin!", "My name is Robin"]
    },
    {"tag": "weather",
     "patterns": ["How's the weather", "What's the temperature?", "How hot is it?", "how cold is it"],
     "responses": ["Todays weather is: ", "The weather is: "]
    },
    {"tag": "location",
     "patterns": ["where are you from", "Where do you live?"],
     "responses": ["I'm from Zwijnaarde.", "I live in Zwijnaarde."]
    },
    {"tag": "hobbys",
     "patterns": ["what are your hobbys", "What do you do in your spare time?"],
     "responses": ["I like to go for a run or go swimming, I'm also active in a student union."]
    },
    {"tag": "school",
     "patterns": ["where do you go to school", "What are you studying?"],
     "responses": ["I'm a student at Arteveldehogeschool.", "I'm a New Media Development student at Arteveldehogeschool."]
    },
    {"tag": "beer",
     "patterns": ["what is your favorite beer?", "which beer do you like the most"],
     "responses": ["Omer hoe liever.", "You can't got wrong with an Omerken or a Stellaken."]
    },
    {"tag": "pets",
     "patterns": ["do you have any pets", "what kind of pets do you have?"],
     "responses": ["I have a cat, a rabbit and a bunch of fish.", "A cat named Truffel, a bunny named Lukie and lots of unnamed fish."]
    },
    {"tag": "football",
     "patterns": ["Whats your favorite sports team", "Who do you support in football", "what football team do you like the most?"],
     "responses": ["Gent of course!", "KAA Gent obviously"]
    },
    {"tag": "cats",
     "patterns": ["Do you know a cat fact?", "do you know any facts about cats", "do you know a fact about cats"],
     "responses": ["Of course I do: ", "Sure: "]
    },
```

Dit is een deeltje van de json-file. Elk thema waar de bot over kan praten is een json object. Er zijn enkele patterns die worden gebruikt om het model te trainen zodat het gepast kan antwoorden op de gebruiker. De bot kiest dan een random antwoord uit een van de responses.

main.py

```
    for intent in data["intents"]:
        for pattern in intent["patterns"]:
            wrds = nltk.word_tokenize(pattern)
            words.extend(wrds)
            docs_x.append(wrds)
            docs_y.append(intent["tag"])

        if intent["tag"] not in labels:
            labels.append(intent["tag"])

    words = [stemmer.stem(w.lower()) for w in words if w != "?"]
    words = sorted(list(set(words)))

    labels = sorted(labels)
```

In dit stuk worden alle worden alle tags en patterns uit de json-file verwerkt. Met nltk.word_tokenize worden de zinnen uit de patterns gesplitst in de afzonderlijke woorden aan dan aan de words list toegevoegd. Deze words list wordt dan vervolgens via de nltk stemmer teruggebracht tot de stam van het belangrijkste woord uit de zin.

```
training = []
output = []

out_empty = [0 for _ in range(len(labels))]

for x, doc in enumerate(docs_x):
    bag = []

    wrds = [stemmer.stem(w) for w in doc]

    for w in words:
        if w in wrds:
            bag.append(1)
        else:
            bag.append(0)

    output_row = out_empty[:]
    output_row[labels.index(docs_y[x])] = 1

    training.append(bag)
    output.append(output_row)
```

Op dit moment hebben we enkel strings als datatypes, maar om te kunnen werken met het machine learning model hebben we getallen nodig. Daarom worden hier dus de lijsten met woorden omgezet naar lijsten die one-hot encoded zijn. Deze kan het model dan gebruiken om gepaste voorspellingen te maken.

```
tensorflow.reset_default_graph()

net = tflearn.input_data(shape=[None, len(training[0])])
net = tflearn.fully_connected(net, 10)
net = tflearn.fully_connected(net, 10)
net = tflearn.fully_connected(net, 10)
net = tflearn.fully_connected(net, len(output[0]), activation="softmax")
net = tflearn.regression(net)

model = tflearn.DNN(net)

try:
    model.load("model.tflearn")
except:
    model.fit(training, output, n_epoch=1000, batch_size=8, show_metric=True)
    model.save("model.tflearn")
```

Het eigenlijke model met een input layer, 3 hidden layers en een output layer. De inputlayer heeft de lengte van het aantal woorden. De hidden layers zorgen voor de eigenlijke voorspellingen van het resultaat. Dit wordt dan doorgegeven aan de output layer die op basis van de hidden layers voor elke mogelijkheid een waarschijnlijkheid geeft.

```python
def chat(inp):
    while True:
        results = model.predict([bag_of_words(inp, words)])
        results_index = numpy.argmax(results)
        tag = labels[results_index]


        for tg in data["intents"]:
            if tg["tag"] == tag:
                responses = tg["responses"]

        if tag == "weather":
            return random.choice(responses) + calls.get_current_weather() + " °C"

        if tag == "cats":
            return random.choice(responses) + calls.get_cat_fact()

        return random.choice(responses)
```

De eigenlijke chatfunctie van de bot. De input van de user wordt gebruikt om via numpy.argmax de meest waarschijnlijke optie terug te geven.

server.py

```python
from flask import Flask, render_template, request
import main

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/get')
def get_bot_response():
    userText = request.args.get('msg')
    return main.chat(userText)

host = '192.168.0.187'
port = 8080
if __name__ == '__main__':
    app.run(host=host, port=port, debug=True)
```

Het servergedeelte van de applicatie. Door gebruik te maken van flask en render_template is het zeer makkelijk om een dashboard te creëren voor de gebruiker. Via request wordt de input van de gebruiker doorgegeven aan de chatfunctie uit main.py.

# 5. Deliverables

## Handleiding

Ik ben begonnen met het maken van de json-file. Hiervoor heb ik een aantal onderwerpen gekozen waarover je de bot kan vragen. Voor elk thema heb ik een aantal mogelijke vragen bedacht die de gebruiker zou kunnen stellen. Ook heb ik elk thema voorzien van een twee à drietal mogelijke antwoorden waaruit random wordt gekozen zodat het iets dynamischer lijkt.

Vervolgens heb ik de logica voor de bot geschreven. Hiervoor heb ik eerst enkele modules geïnstalleerd: ntlk, tensorflow, tflearn en numpy. Eerst moet de data die aan het model wordt getoond bij de training worden verwerkt. De woorden uit de zinnen moeten worden omgezet in een lijst van enkel 0 of 1, one-hot encoded dus. Daarna wordt het model getraind met de voorbereide data. Ten slotte kan het model antwoorden op input van de gebruiker op basis van de mogelijkheid met de hoogste waarschijnlijkheid. Ter aanvulling heb ik een file gemaakt met enkele API calls aan de hand van requests.

Om een online dashboard te maken voor de bot heb ik gebruik gemaakt van flask. Hiervoor heb ik een html-template gemaakt waarmee de gebruiker kan communiceren met de bot. Wanneer de gebruiker iets typt, wordt dit doorgegeven aan de chatfunctie van de bot en die geeft een zo accuraat mogelijk antwoord.

## Timesheet

Voor ik aan het project begon heb ik eerst heel wat opzoekwerk gedaan over hoe ik het best een chatbot zou kunnen implementeren. Nadat ik op de manier uitkwam die ik uiteindelijk heb gebruikt, heb ik mij eerst ook nog wat verdiept in de basis van machine learning aangezien dit toch wel belangrijk is om te begrijpen bij deze bot.

Vervolgens heb de main functionaliteit van de bot geschreven. Deze heb ik eerst enkel in de console laten werken. Hierbij ben ik een aantal bugs tegengekomen, maar deze heb ik vrij snel kunnen wegwerken eigenlijk.

Daarna ben ik begonnen aan het online dashboard van de bot. Hiervoor heb ik eerst ook nog wat opzoekwerk gedaan om te weten hoe ik dit het best kon uitvoeren. De uitwerking hiervan ging zeer vlot en zonder problemen.

Dan heb ik nog enkel API calls toegevoegd om de bot wat interessanter en dynamischer te maken. Hierbij heb ik ook niet echt problemen ondervonden.

De laatste stap was om de bot te koppelen aan de Sensehat van de Raspberry Pi. Zoals eerder al vermeld is dit nog niet gelukt op het moment van deze deadline wegens een heel aantal problemen. Ik moet wel eerlijk toegeven dat ik dit project een beetje had laten liggen en ik mezelf dus wat in tijdsnood heb gebracht.