



UNSW

THE UNIVERSITY OF NEW SOUTH WALES

SCHOOL OF MECHANICAL AND MANUFACTURING ENGINEERING

COMP 9900

Computer Science / Information Technology Project

Group: Mr. Robot (Final Report)

Xiang Zhou	z5147351
Mengyi Shi	z5104528
Zheng Lu	z5135652
Sikai Huang	z5156635

Contents

1. Introduction	4
1.1. Incentive	4
1.2. Goal	4
1.3. Report Structure.....	4
2. Background.....	5
2.1. Usage Scenario	5
2.1.1. Log in.....	5
2.1.2. IOT	5
2.1.3. Weather.....	5
2.1.4. Flight Booking.....	5
2.1.5. Music	5
2.1.6. Context Switch.....	6
2.1.6.1. Weather <=> Flight booking.....	6
2.1.6.2. Artist <=> Album	6
2.1.7. Context resume	6
2.2. Architecture	8
2.2.1. Architecture Overview.....	9
2.2.2. Architecture Request Flow	9
3. Main Component 1- Face Recognition:	10
3.1. Architecture Overview.....	11
3.1.1. Training Phase	11
3.1.2. Recognition Phase	12
3.2. Technical Discussion	12
3.2.1. Affine Transform.....	12
3.2.2. Fine Tune SVM:	13
3.2.3. Stricter Policy:	13
3.2.4. Increase the training sample for SVM:	13
4. Main Component 2- Context Switch:	13
5. Main Component 3- IOT and Service Interaction:	15
5.1. Flux smart light bulb:.....	15
5.2. Auto dog feeder:	15
5.3. Other Service API:.....	16
5.3.1. Weather API	16
5.3.2. Music API.....	16
6. Conclusion.....	17
7. Appendix	18

- 7.1. Used Technique 18
 - 7.1.1. Frontend..... 18
 - 7.1.2. Backend 18
- 7.2. Backend Code Structure: 18
- 7.3. Installation Manual 19
 - 7.3.1. Open Node Command Prompt window 19
 - 7.3.2. Change Directory to /chat-bot/chat-front-end..... 20
 - 7.3.3. Type in “npm install” to install the frontend..... 20
 - 7.3.4. Type in “npm start” to start the frontend 20
 - 7.3.5. Contact E-mail Address 21

1. Introduction

1.1. Incentive

The project of our group aims to build a butler chat bot that are not only user friendly but also more versatile than any other existing home AI including Google Home, or Alexa and Echo from Amazon, or Home Pod from Apple, etc. However, they are more focused on functionalities that relies heavily on their own product line. Besides, their insufficient attention on IOT equipment introduces a limitation to the usability of the system. Our chat-bot, though not developed as they are, provides a new angle on how we can implement a house butler AI accustomed to most users and integrate services provided by a variety of companies as well as devices together, to achieve a better performance. In our implementation, to solve the imbalance of functionalities discovered in the previously mentioned systems and to develop versatility, we integrated multiple APIs from different sources to achieve different functionalities including music, weather, and IOT. Besides that, our effort in improving usability is worth mentioning. We applied a face recognition procedure during the log in process not only to facilitate the login process (that way the user never have to memorize and enter their credential every time they use the system), but also enhance the security by significantly increasing the difficulty of performing an “impersonation attack”.

1.2. Goal

Our aim is to provide a chatbot shouldering the responsibility of a butler to organize the daily life of its users through conversation effortlessly. It is focused on properties as follows:

- Usability
- Diverse Functionality
- Security
- Customizability

However, due to the limitation of developing time, our ideas are merely realized by showing an example in every aspect mentioned above.

1.3. Report Structure

The report will first introduce the background of the system, including usage scenarios and system architecture. Then 3 main components will be illustrated in detail, with codes and workflow demonstration. After technical details, a brief evaluation on the development process of the system will be given, along with expectations on potential improvement for the system. The appendix of the report contains the technical profile of the system, an installation manual and a source code structure.

2. Background

2.1. Usage Scenario

This part mainly describes the scenarios we expect the users to use the system. As a house butler chatbot, the potential users of the system are ordinary people who are interested in having an intelligent home. The system will co-operate with multiple IOT devices as well as online services to achieve the functionalities we expected.

2.1.1. Log in

Press “Login” => Camera invoked => Photo taken (then sent to backend, examined against the data in database) => user recognition successful => login successful

2.1.2. IOT

Input: Turn on the light/light it up

Output: Lightbulb lit up Input: Turn off the light/shut the light

Output: Lightbulb turned off

2.1.3. Weather

User: What is the weather?

System: What is the date?

User: Today

System: Which city?

User: Sydney

System: Sunny, 20 degree, etc.

2.1.4. Flight Booking

User: I want to book a ticket from Sydney to Melbourne.

System: When is the departure time?

User: Tomorrow.

System: You have booked a flight from Sydney to Melbourne tomorrow!●●●

2.1.5. Music

●Play song

User: Play Let It Go.

System: Spotify widget to play song Let It Go.

●Play album

User: Play Eat a Peach

System: Spotify widget to play album Eat a Peach.

●Play album by an artist

User: Play album by Eminem.

System: Spotify widget to play an album of Eminem.

●Play song by an artist

User: Play Taylor Swift.

System: Spotify widget to play a song by Taylor Swift.

2.1.6. Context Switch

●Weather <=> Flight booking

User: I want to book a flight from Shanghai to Brisbane tomorrow.

System: You have booked a flight from Shanghai to Brisbane tomorrow!

User: What is the weather?

System: Weather in Brisbane tomorrow is 18 degree, cloudy, etc.

●Artist <=> Album

User: Play Eagles.

System: Spotify widget to play a song by Eagles. (Artist stored in context)

User: Get an album.

System: Spotify widget to play an album by Eagles. (Used artist in the context)

2.1.7. Context resume

User A login.

User A: What is the weather in Sydney tomorrow?

System: Weather in Sydney tomorrow is 22 degree, showers, etc. (Context set. Address and date stored in the context)

User A log out. (Context stored into Database)

User B log in and input a series of requests. Then user B log out.

User A log in. (Context loaded from Database)

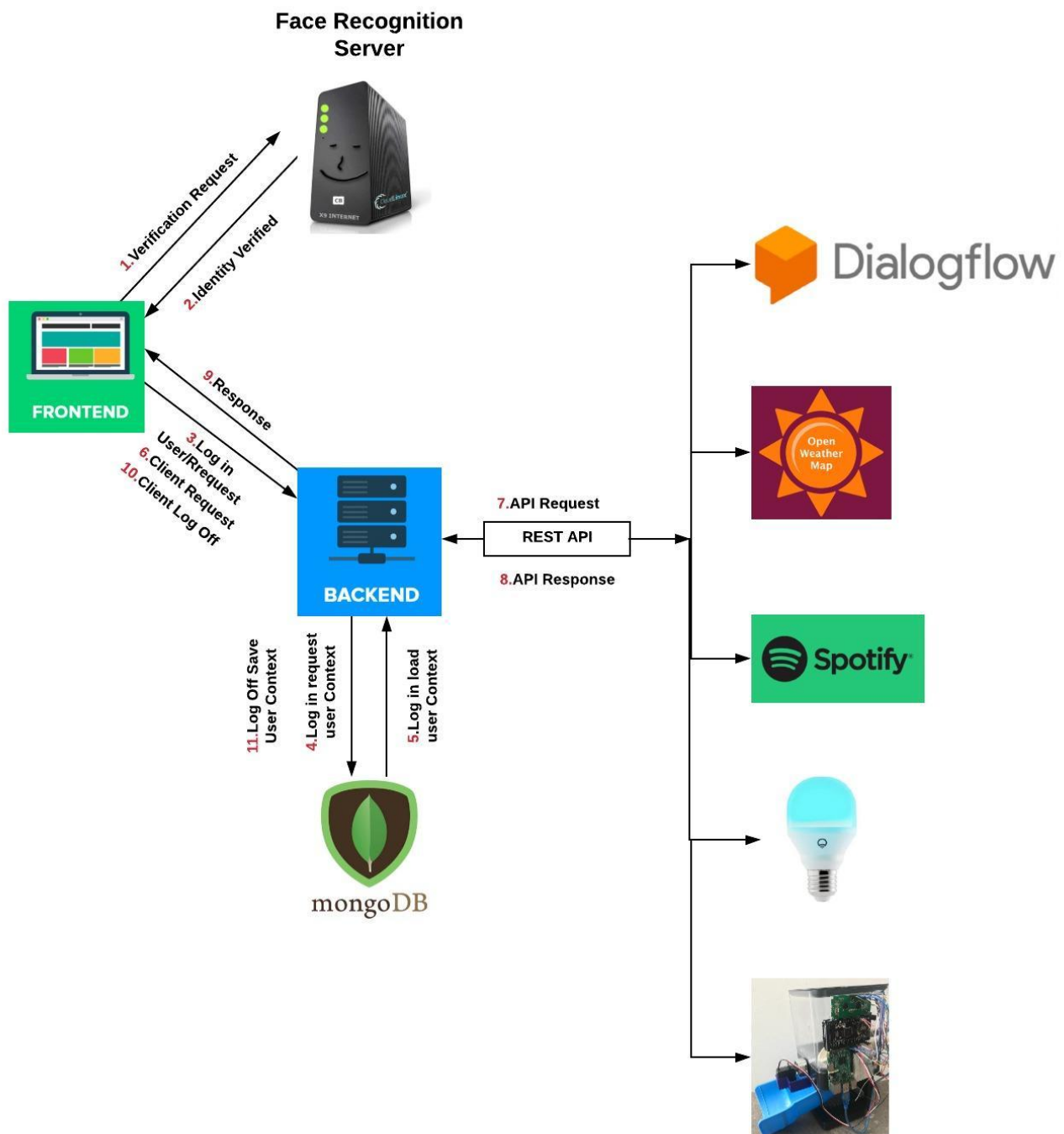
User A: I want to book a flight.

System: From which city?

User A: Perth.

System: You have booked a flight from Perth to Sydney tomorrow! (Used address: Sydney, date: tomorrow from the context set before the previous logout).

2.2. Architecture



2.2.1. Architecture Overview

We have a server running at the front end at the client side and 3 other servers running at the same time. The three servers are:

- Backend: which is the central unit of this architecture, that will process the incoming request and then send request to the API according in the diagram and process the response and send to the front end.
- Face Verification Server: will process the incoming image of the logging user and send back the user identity.
- Mongo DB: Will retrieve and save user context at log in and log off time respectively.

The other components are the API resources that is available for backend to enquiry upon, and they are services including:

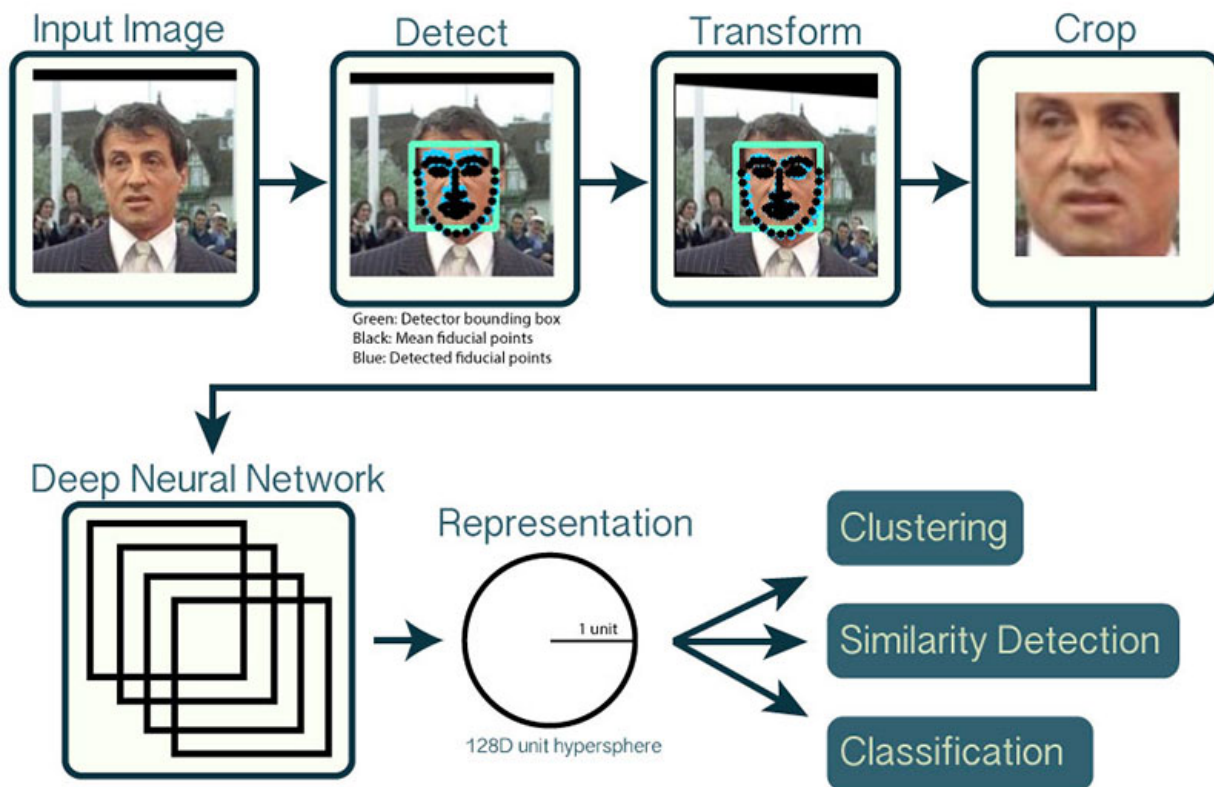
1. Spotify music API
2. Open weather map API
3. Dialog flow API
4. IOT interaction API (smart light bulb)
5. IOT interaction API (auto dog feeder)

2.2.2. Architecture Request Flow

As shown in the diagram below are the explanation of the flow of the request:

1. User will be asked to log in with their faces in the front end and after user send the image of their face at the front end, a verification request will be sent to the face recognition server at the face recognition server.
2. The face recognition server will process the image and return to the front end the identity of the user
3. After the front end get the user identity, it will send a login request to the backend
4. When Backend received a login request from the from end, it will issue a load user context to the database
5. The database will retrieve the user context and return to the backend
6. Now the log in process has finished and the front end will send client request to the backend
7. The backend will check user request and will enquire the relevant API service accordingly, for example backend will send the query first to the dialog flow and find out that the user want some music and it will then send request to the music API with the relevant parameters.
8. The service API will send the backend a response
9. After processing the response, the backend will return the response to the front end
10. When user request to log off, the front end will send the log out request to the backend.
11. Backend will the save the user current context to the database to next log in retrial

3. Main Component 1- Face Recognition:



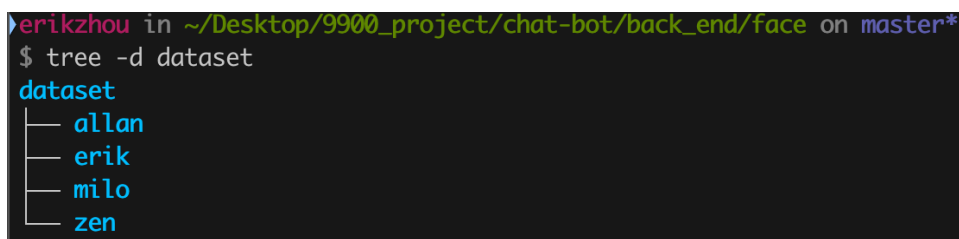
3.1. Architecture Overview

As shown in the face recognition diagram above, these are the steps to processing face recognition: When received an image, the server will first detect where the face is, using Caffe Model for face detection. With the face detected, it will then crop the image with just the face and then it could perform affine alignment, which means it will find the landmark of the face (eyes, and mouth) and align the landmark so that the face is right in the middle not tilted. An example will be shown later. With the face image we have just detected, we will resize the image into (96 x 96 x3 since the CNN image input size needs this) we then put it into the face recognition CNN neural network using the open face pretrained face recognition model and it will then output 128 encoding of the image. Then on top of the output of the CNN for face detection (128 encoding for each image) we could do a clustering algorithm for each image and in our case, we choose SVM.

3.1.1. Training Phase

We have collected 50 images of each of the team member in different light conditions and saved them in:

chatbot/back_end/face/dataset



```
erikzhou in ~/Desktop/9900_project/chat-bot/back_end/face on master*
$ tree -d dataset
dataset
├── allan
├── erik
├── milo
└── zen
```

In the dataset folder each team member has a separate folder with their image saved in the named folder.

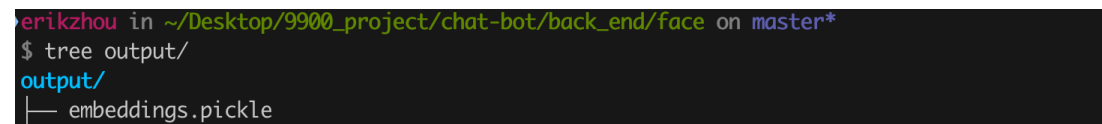
Now we have the data and we need to put all images to the CNN and get the 128 encodings together saved with its labels (which person this encoding belongs to), so we will have for each image its 128 encoding (output from CNN) and its label (the person of this image) for the classification algorithm SVM.

This has been implemented in:

chat-bot/back_end/face/extract_embeddings.py

The labelled data will be saved as pickle file in:

chat-bot/back_end/face/output/embeddings.pickle



```
erikzhou in ~/Desktop/9900_project/chat-bot/back_end/face on master*
$ tree output/
output/
├── embeddings.pickle
```

Then we need to be using this labelled data for training SVM:

chat-bot/back_end/face/train.py

It will use the data in embedding. Pickle and train a classifier and save the trained classifier in:

chat-bot/back_end/face/output/recognizer.pickle

3.1.2. Recognition Phase

The recognition function is implemented in:

chat-bot/back_end/face/network_image.py

```
(carND-term1) erikzhou in ~/Desktop/9900_project/chat-bot/back_end/face on master*  
$ python network_image.py -d face_detection_model -r output/recognizer.pickle -l output/le.pickle
```

issuing the above command will start the face recognition server at port 5000

The incoming image will be first detected and cropped where the face is and then the face image will be resized to (96,96,3) in order to pass in to the CNN to get the 128 encoding of this image. Then we pass the 128 encoding in to the classifier that we have previously trained and get the output of the identified user.

When there is an incoming image send from the front end, it will process and send the front end the identity of the person and below are the debug outputs from this code:

```
127.0.0.1 - - [26/Apr/2019 22:18:53] "OPTIONS / HTTP/1.1" 200 -  
{'erik': 0.9148309374318426}  
[Info] erik is recognised with prob of 0.9148309374318426  
[Info] This return to the front end: {'user': {'userID': 1, 'userName': 'erik'}}  
127.0.0.1 - - [26/Apr/2019 22:19:06] "POST / HTTP/1.1" 200 -
```

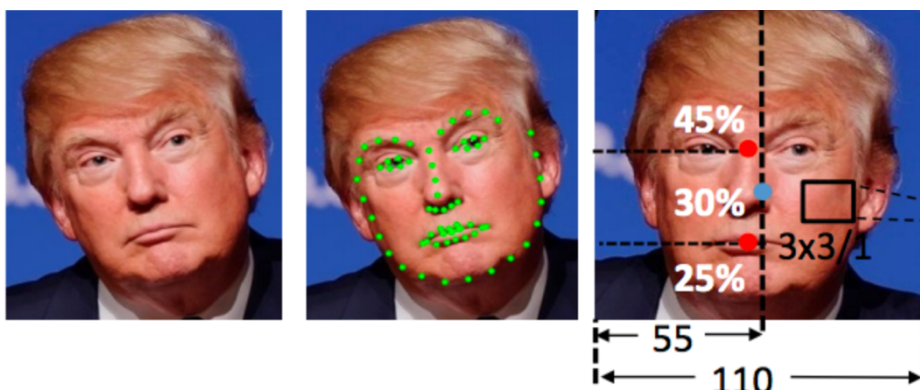
As it can be seen in this case, I have been identified correctly with confidence of 91% and it will send to the front end my user name and id.

3.2. Technical Discussion

During the implementation there are couple of measurements that has been taken into account to improve the accuracy of the face detection

3.2.1. Affine Transform

Using affine transform to align the face. As it has been explained previously this method will first detect the landmark of the face(eyes and mouth) and they should form a triangle and use this as our reference points we could perform affine transform to the image so that the face is aligned just right in the image and below is a demo of this method:



We have tried to implement this method and it turns out it is good at some cases where the user face is not aligned properly and the accuracy improves only slight, but we have a huge performance penalty since for each frame that we have detected a face , we need to perform such transformation and it is computational

expensive. Since it is magnitude slower than the method that is without affine transform but with only slight improvement of accuracy, we decide not to use this method.

3.2.2. Fine Tune SVM:

The reason that we choose SVM for our classifier algorithm is because it performs well with only small amount of data. However, to achieve an accuracy result, the hyper parameters tuning has some subtleties. We have tried different parameters for SVM, and it turns out that $C = 1$, kernel = "linear" will gives us the best result as show in the code snippet below:

chat-bot/back_end/face/train_model.py

```
34 recognizer = SVC(C=1, kernel="linear", probability=True)
```

3.2.3. Stricter Policy:

We have also considered that by Applying more restriction for identify user will supress the false positive result. So not only we will need to compare the result of the SVM confidence we will also compare the image encoding of that identified person with all the encoding in the data base with the person recognised and compute the smallest Euclidian distance between the encodings, if this difference is larger than a threshold we will say that we could not recognise this image.

For example, if there is an incoming image being identified as Erik, then it will go to the data base with all the encoding stored with Erik and compare the encoding with the encoding with this incoming image and record the smallest distance of the encodings. Then we will decide if we reject this recognition base on both the SVM confidence as well as the distance, if the distance is larger than a threshold then we could say that this person does not exists in our database.

The implementation is in:

chat-bot/back_end/face/face_strick.py

However, in practice this will slight supress the false positive examples, but it is much more stick in terms of asking user to be in better light condition and with face aligned properly, otherwise user will get easily rejected. Our project does not use this implementation, but it is an option for suppressing the false positive but with Stricker input requirement.

3.2.4. Increase the training sample for SVM:

Initially we only collected about 10 images for each person and the accuracy is not very good and when we get 50 images for each person the accuracy improves significantly. We have collected 50 images of each person from different lighting and then randomly choose 5 images out of the 50 images as test set. It turns out that by increasing data training set to 50 images per person for SVM will improve the accuracy to above 95%.

4. Main Component 2- Context Switch:

As mentioned before, in our application when user login the backend will request from the database to retrieve the user's previous context so that user would continue their last conversation without out the chat bot prompting user for the same information that he has already mentioned and when user log off, we need to save

the user's current active context to the database for his next log in retrieval. The implementation has been done in:

chat-bot/back_end/back2diag.py

```
129 def save_user_context(userid):
130     context_list = [] #google.cloud.dialogflow_v2.types.Context
131     for e in context_client.list_contexts(parent):
132         print("=====")
133         print("[Info] Active Context:")
134         print(e)
135
136         pay_text = MessageToJson(e) #change the google class to string
137         context_list.append(pay_text)
138
139
140     #now serialise the list and save to database
141     s_list = pickle.dumps(context_list) #list serialisable
142     update_user(str(userid),"content",s_list) #save the context to database
```

In the code above, this function is called when user log off and it will get the contexts from the dialog flow and then serialise the context list using pickle. In this function we have utilised google protocol buffer tool MessageToJson() (line 136) to serialise this google context class object(more on this later).

```
101 def load_user_context(userid):
102     new_context = get_context(str(userid))
103     if new_context == "":
104         return False
105
106     print("[Info] The load user context list from database:",new_context)
107     context_list = pickle.loads(new_context) #change string to list
108     for s in context_list:
109         data = json.loads(s) #make each context string to json
110         print("[INFO] The data is:",data)
111         name = data["name"]
112         if "lifespanCount" in data:
113             life = data["lifespanCount"]
114         else:
115             life = 1
116
117
118     #create a template context class
119     temp = {"name":name,"lifespan_count":life} #a template to create context
120     blank = context_client.create_context(parent,temp) #create a blank context
121     context_client.delete_context(name) #delete the previous temp
122
123     #restore the context
124     restore = Parse(s,blank)
125     result = context_client.create_context(parent,restore) #create a blank context
126
127     return True
```

On the other hand, when user log in, above function will be called to load the user context. We first de-serialise the context list from pickle file (line 107) then we go through each serialised context string within the context list and retrieve the context name and lifespan_count for each context(line108-115). After that we use the context name and lifespan_count to create a temporary context as a template for later de-serialise the context class.

Since the context is actually of:

```
<class google.cloud.dialogflow_v2.types.Context>
```

We need special tool google protocol buffer in order to deserialize such class. The relevant python modules have been imported as below:

```
37 from google.protobuf.json_format import MessageToJson
38 from google.protobuf.json_format import Parse
39 from google.protobuf.struct_pb2 import Struct, Value
```

The method Parse () becomes handy in this case, it will take two arguments, one is the serialised text string of the context class object, which we have already retrieved from the database(line 108) the other one is the actual class of its type as a template to deserialize this special object, which we have already created (line 119). Finally, once we got the context object successfully deserialize, we could load the context to the dialog flow.

5. Main Component 3- IOT and Service Interaction:

5.1. Flux smart light bulb:



The Backend is able to interact with smart IOT devices, in this case we use Flux smart light bulb. The light bulb would be initially configured on the mobile device within the same network and then send the authentication token to the backend. The cloud will register the light bulb IP location and authentication token, then each time there is a request from the user to control the light, the backend will send the request to the cloud server with the authentication token and after verifying the authentication token the cloud server will then send the request to the light bulb.

5.2. Auto dog feeder:



We also can control the dog feeder through the chatbot, and the communication architecture is as above. The backend will send a http request to the raspberry pi server which will listen to the dog feeding request and then send pwm signal to the dog feeder servo to control the motor that will spin and start feeding the dog. The implementation is in:

```
chat-bot/back_end/api_service/dog_feeder/dog.py
```

```
1 import RPi.GPIO as GPIO
```

We used RPi.GPIO module to control the raspi pin output to control the motor, that is to change the duty cycle of the pwm output to control the degree that the servo is turned.

```
16 duty = angle/ 18 + 2
17 GPIO.output(3,True)
18 pwm.ChangeDutyCycle(duty)
```

5.3. Other Service API:

5.3.1. Weather API

We used open weather API to implement the service related to weather query. The implementation has been done in:

chat-bot/back_end/api_service/weather/weather_api.py

This API contains one function which requires two parameters the city name and time:

```
def get_forecast(city_name,when):
```

After filter the date of weather user wants and transform the temperatures unit to degree Celsius, a json data will be transferred to the helper module and then response to the user.

5.3.2. Music API

The music API part is mainly based on Spotify music API. As for Spotify has implemented many useful recommendation and query logic, we can simply get the login token and make the calls. We allow the user to search an album or track in the conversation and the a embed URL of such a resource will be return to the front-end module to insert a play window into the conversation widget. What is more, the music API module also contains the function to show one recommendation track of an artist. The function is below:

```
def show_recommendations_for_artist(name):
    refresh_sp()
    artist = get_artist(name)
    content = []
    results = sp.recommendations(seed_artists=[artist['id']])
    #print(results)
    for track in results['tracks']:
        # check if search the specific name of the artist by id
        if(artist['id'] == track['artists'][0]['id']):
            external_url = track['external_urls']['spotify'].replace('com', 'com/embed')
            #print(track['name'], '-', track['artists'][0]['name'])
            content.append({'name': track['name'], 'url': external_url, 'artist_name': track['artists'][0]['name']})
    #data = json.dumps({'type': 'track', 'contents': content})

    #select random song from content
    data = {'type': 'track', 'contents': random.choice(content)}

    #print(data)
    return data
```

All functions will return a json data which contains the 'track(album) name, URL, artist_name'.

The implementation has been done in:

chat-bot/back_end/api_service/weather/spotify_api.py

The path of the data source is:

chat-bot/back_end/database/data/user.json

Run ‘*mongod -dbpath address*’ to set the path of the database and run :

mongoimport -d chatbot -c users --file ./users.json

to initial the data.

Basically, the create and update methods are implemented in:

chat-bot/back_end/database/userservice.py

6. Conclusion

In our previous description of various drawbacks in systems already in the market, we mentioned not only the imbalance in functionalities but also the lack of hardware involvement that limits the performance and the usability of the systems.

In our approach to these concerns, we considered integrating a wide range of commercial APIs as well as different IOT devices into the system to better achieve our goal. First, we successfully enable the system to read the intention of the user through conversation with aid from Dialogflow. On the other side, for APIs we included a weather service (Openweather) and a music service (Spotify) so that the chatbot can provide weather information and music playing service to the user. Also, we managed to implement a controlling interface on a smart lightbulb and an automatic-dog-feeder to showcase the ability of the system to control different household equipment. Besides that, as an improvement on usability and security of the system, we made use of opencv library to achieve face recognition in login process with a considerable high accuracy and acceptable efficiency.

In terms of potential development, we can perfect the system in different aspects. For example, to put on more versatility, we consider finding more services to add more functionalities in providing information as well as performing on-line actions (including takeaway ordering, hotel reservation, or even mail reception/composing). Additionally, with the trending of smart device on the market, we can expect the system to have control of all the equipment at home (of course we need to consider security at this stage) including lighting, television, air conditioner, or even the stove so that the user can save the effort of manually set up and configure devices to use them. It is also possible to construct an auxiliary database storing the preferences of the user to accustom the system to the user as an improvement in the usability.

7. Appendix

7.1. Used Technique

7.1.1. Frontend

Used **React** framework with **Ant design** as a UI aid during development to render frontend chat widget.

7.1.2. Backend

Python Flask framework used in the implementation of the backend routing logic.

●Face recognition

Applied **People Detection Model** from *Caffe* to detect human face and **Openface CNN Pre-train Model** implement face recognition.

●Vacuum Cleaner

Used **MQTT Protocol** to control a vacuum cleaner.

●Auto Dog-Feeder

Used **Raspberry Pi** to receive command from backend and **PWM** control the motor operating the motor of the machine.

●Context Switch

Used **MongoDB** to store the active context when the user logs out. **Google Protocol Buffer** in serializing contexts.

7.2. Backend Code Structure:

•*chat-bot/back_end/*back2diag.py

This file is where the backend has been implemented (corresponding to the backend in the Project architecture, which would take control of all the coming front-end traffic and coordinate all the services APIs on and dialogflow NLP.

- chat-bot/back_end/helper.py***

Define some of the helper functions in the **back2diag.py** to call API services

- chat-bot/back_end/face/network_image.py***

Is the implementation of the face recognition server

- chat-bot/back_end/face/face_lib.py***

An implementation library for ***network_image.py*** to implement face recognition

- chat-bot/back_end/face/extract_embeddings.py***

Given an input of image will output the 128 encoding of the image using open face CNN pre trained model.

- chat-bot/back_end/face/train_model.py***

Using the encoding and label that outputted by ***extract_embeddings.py*** train the SVM for face classifications.

- chat-bot/back_end/face/inception_blocks_v2.py***

The open face CNN architecture for encoding the face image

- chat-bot/back_end/face/fr_utils.py***

Utility functions that used for face recognitions

- chat-bot/back_end/api_service***

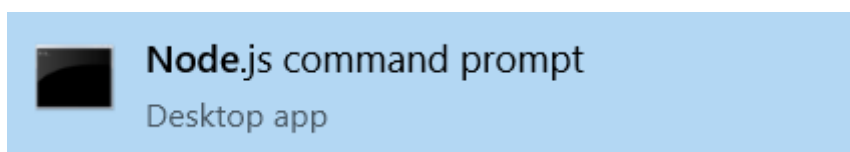
All the API services functions and utilities

7.3. Installation Manual

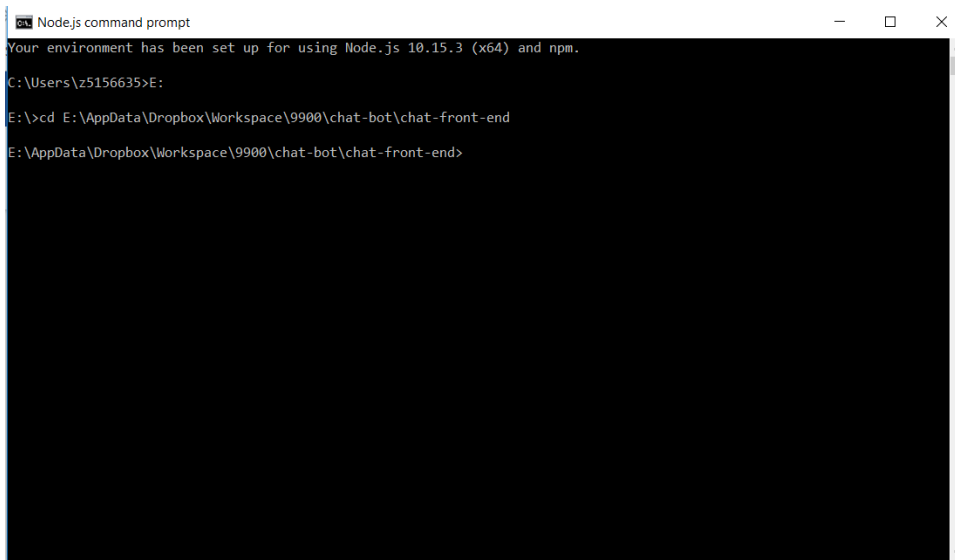
Since we have our backend deployed on AWS, the only installation we need to start the system is the installation of the frontend on the user's machine. As per spec we assume the OS on the machine is Windows 10. The steps are:

7.3.1. Open Node Command Prompt window

Press “Windows” button and search for “Node command prompt”.



7.3.2. Change Directory to /chat-bot/chat-front-end



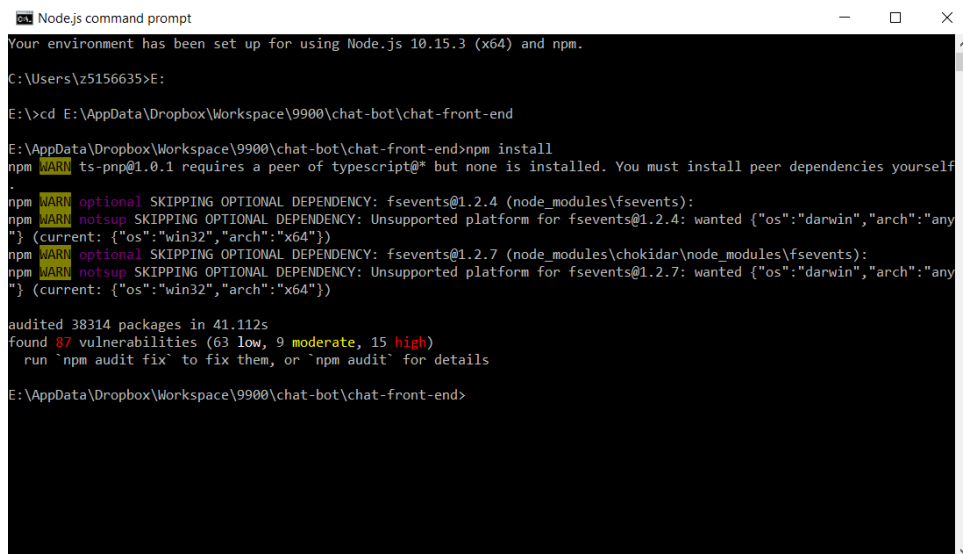
```
Node.js command prompt
Your environment has been set up for using Node.js 10.15.3 (x64) and npm.

C:\Users\z5156635>E:

E:\>cd E:\AppData\Dropbox\Workspace\9900\chat-bot\chat-front-end
E:\AppData\Dropbox\Workspace\9900\chat-bot\chat-front-end>
```

7.3.3. Type in “npm install” to install the frontend

Allow a few minutes before the installation finishes.



```
Node.js command prompt
Your environment has been set up for using Node.js 10.15.3 (x64) and npm.

C:\Users\z5156635>E:

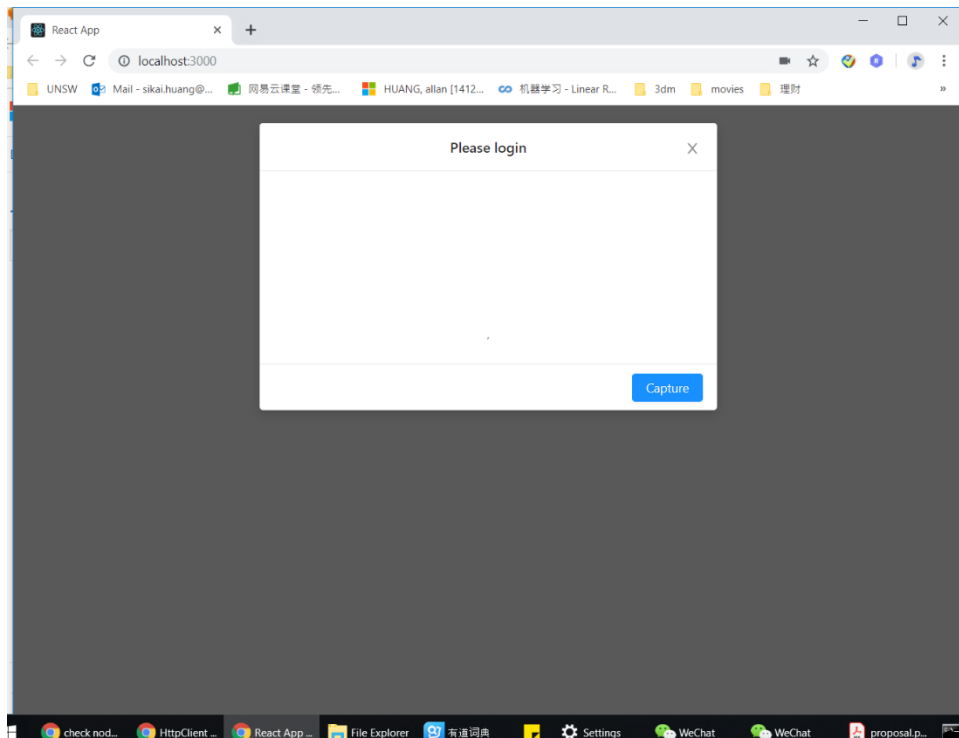
E:\>cd E:\AppData\Dropbox\Workspace\9900\chat-bot\chat-front-end
E:\AppData\Dropbox\Workspace\9900\chat-bot\chat-front-end>npm install
npm WARN ts-pnp@1.0.1 requires a peer of typescript@* but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.7: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

audited 38314 packages in 41.112s
found 87 vulnerabilities (63 low, 9 moderate, 15 high)
  run `npm audit fix` to fix them, or `npm audit` for details

E:\AppData\Dropbox\Workspace\9900\chat-bot\chat-front-end>
```

7.3.4. Type in “npm start” to start the frontend

A webpage will automatically be opened in the default browser and connects the backend running on AWS.



Then the user may log in using the face recognition.

7.3.5. Contact E-mail Address

If any problem arises, here is the contact details.

E-mail: z5147351@login.cse.unsw.edu.au.

Contact Person: Erik Zhou (Scrum Master)