Implementing YouTube Search API with WebClient

SUSI.AI is an assistant which enables us to create a lot of skills. These skills offer various functionalities which performs different actions. Therefore SUSI has implemented various action types. Some of these are:-

- Answer
- Table
- Maps
- Rss
- audio_play
- video_play

When a user answers a query the server sends response in form of actions type. The client then scans the response JSON object and checks the type of action and performs the desired operations. This action types are verified by MessageListItem.react.js which creates the answer messages and displays them on the client. The server looks in the expert files and find the correct skill from susi_skills_data. These skills defines the rules for the answers to be send by the user in form a json object. Given below is an example to a skil to search for a youtube video ID :-

```
play * on youtube
!console:
{
  "url":"https://www.googleapis.com/youtube/v3/search?part=snippet&q=$1$&key=apikey",
  "path":"$.items[0].id.videoId"
  "actions":[{
          "type" : "video_play",
          "identifier_type" : "youtube",
          "identifier": “$object$”
  }]
}
eol
```

whenever a user sends a query which is similar to the above query with a word or group of words between “play” and “on youtube” the words between them will be stored in a variable 1 and the complete query is stored in 0.
Below the query there is an object with has three attributes : url, path, and actions.
The url contains the value to the request url to youtube data v3 api server.

```
"https://www.googleapis.com/youtube/v3/search?part=snippet&q=<search term>&key=<your api key>”
```

This HTTP request can contains multiple queries, here only two are applied. First the search term which is set to variable q=<search term here> and second the key which is set to the api key of the developer issued by google at your console at console.developers.google.com

The YouTube server sends an API response in form of a json object. Given below is an example of the the response by YouTube api for search of despacito.

```
{
  “items”:{
        0:{
           eta : ”some value object”,
           Id : {
                    Title: “Luis Fonsi - Despacito ft. Daddy Yankee”,
                    VideoId: “kJQP7kiw5Fk”
           }
           data : {
               thumbnail:{some json object}
               descryption:{some json object}
           }
        }
        1:{
           eta : ”some value object”,
           Id : {
                    Title: “Justin Bieber – Despacito (Lyrics) ft. Luis Fonsi & Daddy Yankee”,
                    VideoId: “whwe0KD_rGw”
           }
           data : {
               thumbnail:{some json object}
               descryption:{some json object}
           }
        }
        2:{

        }
  }
}
```

We see an array of items with index from 0 to 5 all of which are the data about the top 5 searched video for our search term. This json file contains all the information about these videos including id, title, description, thumbnail links etc. Each video has a unique video id which can be used to play the video directly. In fact the url any video contains this unique id. for example:

```
https://www.youtube.com/watch?v=kJQP7kiw5Fk
```

where the string after the “?v=” is the video Id of a particular video. Here we see that the string “kJQP7kiw5Fk” is same as the videoId parameter of the first items object of the response json object of query Despacito.

Therefore from the skill rule above we see that this video id of the first item of the response object is passed to the $object$ variable. This $object$ variable is then set to the value to identifier attribute of the actions object. Here is the snippet of the code:

```
"actions":[{
        "type" : "video_play",
        "identifier_type" : "youtube",
        "identifier": “$object$”
}]
```

we see that the type of this actions object is video_play and the identifier_type is “youtube”.

This defines the rule for your answer to the query.

Here is an example of the json object send by susi server to the web client at the api.susi.ai endpoint.


```
{
  "query": "open the pod bay door",
  "count": 1,
  "client_id": "aG9zdF8xNDEuMTAxLjEwNC4xNjNfMTdmYzkxYzU=",
  "query_date": "2018-05-23T10:13:37.820Z",
  "answers": [{
    "data": [{
      "actions": [{
        "type": "video_play",
        "identifier_type": "youtube",
        "identifier": "7qnd-hdmgfk"
      }],
      "0": "open the pod bay door",
      "token_original": "the",
      "token_canonical": "the",
      "token_categorized": "the",
      "timezoneOffset": "-330",
      "language": "en"
    }],
    "metadata": {"count": 1},
    "actions": [
      {
        "type": "video_play",
        "identifier_type": "youtube",
        "identifier": "7qnd-hdmgfk"
      },
      {
```

```
      "language": "en",
      "type": "answer",
      "expression": "I'm sorry, Dave. I'm afraid I can't do that."
    }
   ],
   "skills": ["/susi_skill_data/models/general/Games, Trivia and
Accessories/en/hal_persona.txt"],
   "persona": {}
  }],
  "answer_date": "2018-05-23T10:13:37.943Z",
  "answer_time": 123,
  "language": "en",
  "session": {"identity": {
   "type": "host",
   "name": "141.101.104.163_17fc91c5",
   "anonymous": true
  }}
}
```


This json object is then received by the messageListItem component of the webclient chat.susi.ai.

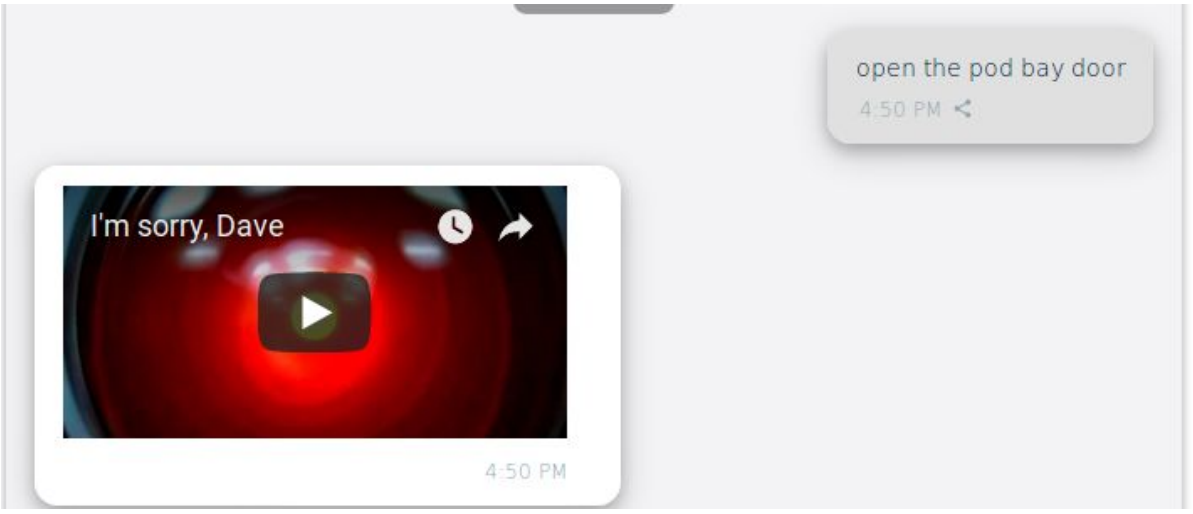
How the client responds to the video_play action

The file MessageListItem.react.js file works like a factory which produces all the different types of messages send by susi. It verifies each action types and performs the requested task. Below is the snippet of code from this file:

```
      case 'table': {
        let columns = data.answers[0].actions[index].columns;
        let count = data.answers[0].actions[index].count;
        let table = drawTable(columns,data.answers[0].data,count);
        listItems.push(
          <li className='message-list-item' key={action+index}>
            <section className={messageContainerClasses}>
            <div><div className='message-text'>{table}</div></div>
              {renderMessageFooter(message,latestUserMsgID,showFeedback)}
            </section>
          </li>
        );
        break
      }
      case 'video_play': {
          console.log(data);
        let identifierType = data.answers[0].actions[index].identifier_type;
        let identifier = data.answers[0].actions[index].identifier;
        let src = `https://www.${identifierType}.com/embed/${identifier}?autoplay=1`;
        listItems.push(
          <li className='message-list-item' key={action+index}>
            <section  className={messageContainerClasses}>
            <div className='message-text'>
                <iframe src={src}
                  frameBorder="0"
                  allowFullScreen>
                </iframe>
              {renderMessageFooter(message,latestUserMsgID,showFeedback)}
            </div>
            </section>
          </li>
```

From the above snippet we see the case ‘video_play’. In this case we have three variable identifierType, identifier and src. First two are received from the response data and are used to create the src variable for the iframe. We can see that the list item with class ‘message-list-item’ is pushed inside the listItems. It contains a div with id message-test class which displays the final answer. It contains an iframe tag which creates the desired Iframe this the src variable created above by the two variables.

The output on screen for the above iframe is below for the query open the pod bay door.



References:

https://developers.google.com/youtube/player_parameters
https://blog.fossasia.org/skills-for-susi
https://blog.fossasia.org/how-to-teach-susi-ai-skills-using-external-apis/