

# FINAL-REPORT

## CS-5368 Intelligent Systems Project Report Spring 2023.

Project: Develop an Intelligent Agent

SUBMITTED BY:

James Ballari (R11836829)

Kuljeet Kaur (R11858474)

Soundarya Samala (R11880448)

Tishya Sohankumar Thakkar (R11840544)

# Table of Contents

---

Introduction .....	3
GitHub Links.....	3
Problem Statement .....	3
Application Domain .....	3
Intelligent Agent .....	3
Problem Description .....	4
Background knowledge .....	4
Questions.....	4
Modeling .....	10
Objects.....	10
Relations .....	10
Knowledge/rules .....	11
Web Application Interface Design .....	12
Input .....	12
Output .....	13
Implementation of Web Application .....	14
Further Details on the above components: .....	14
• The View:.....	14
• The Controller/Logic.....	15
Project Structure .....	22
Techniques .....	23
Web Technologies: .....	23
• HTML .....	23
• JavaScript (jQuery, Ajax).....	23
• CSS(BOOTSTRAP).....	23
ASP: SPARC .....	24
• Data source extraction Hashing required fields using Python.....	24
• Generating relationships using SQLite DB queries .....	24
• Default Values .....	24
GitHub.....	24
References .....	25

# Introduction

---

## GitHub Links

GitHub Repository – <https://github.com/sosamala/CS5368-project-2023Spring>

GitHub Page – <https://sosamala.github.io/CS5368-project-2023Spring/>

## Problem Statement

The goal of the project is to create a GitHub-hosted web application with intelligent behaviour to help customers choose restaurants based on their preferences. The application domain, which is mostly concerned with food and restaurants, requires complex automated reasoning using ASP/SPARC rules. English-speaking or written questions should be understood by the program, and it should be able to react to them. The software's user interface should be simple to use, intuitive to traverse, and appealing to the end user. Recommendations from the app should be based on location, budget limit, and user preferences and preferences. The project's objective is to produce intelligent software that makes personalized restaurant recommendations to individuals based on their needs and interests, making the choice of a restaurant simpler and more enjoyable.

## Application Domain

This project's application domain focuses on restaurants and food. People often struggle to discover the ideal restaurant that satisfies their unique requirements because they have a variety of dietary needs and preferences. The goal of this project is to help users choose the ideal restaurant for their preferences and budget. In order to deliver personalized recommendations, the program will take into account a number of variables, including the preferred location, cuisine, and restaurant budget. This addresses a typical issue that consumers run across when looking for a restaurant that would suit their needs. We aim to solve this problem by creating an intelligent agent that will help the users to select a restaurant by answering their queries.

## Intelligent Agent

The project's intelligent agent is designed to help users choose a restaurant based on their preferences. To give the user tailored recommendations, the agent applies non-trivial automated reasoning through ASP/SPARC rules. The agent has access to a database of restaurants that contains details about each establishment's menu, location, cost, and reviews. Regex will be used by the agent to comprehend user inquiries in English, whether they are spoken or written, and provide answers. After analysing the user's inputs using reasoning techniques, the agent will then make pertinent recommendations based on the user's preferences.

# Problem Description

---

## Background knowledge

There are various restaurants, each with its own ID. There are menu items, a rating, a pricing range, and a category for each restaurant. A restaurant's rating, which ranges from one to five stars, is a score that represents the quality of the food and service the establishment offers. A restaurant's pricing range relates to the cost of the menu items, which can be anywhere between low (\$) and expensive (\$\$\$). The cuisine provided at a restaurant is designated by its category, such as Italian, Chinese, or Mexican. A restaurant's menu will often offer a variety of dishes and beverages, each of which will have a corresponding category, price, and description.

## Questions

Here are some of the questions that you can ask the agent and the expected response:

1. Suggest a restaurant.

Response: Name of a Random restaurant.

Example:

**Messages**

Hey!!!

suggest me a restaurant.

Do you want try "Raising Cane's" Restauraunt today ?

2. Show Top Rated Restaurants

Response: List of top-rated restaurants.

Example:

Show me the Top Rated Restaurants

The Top Rated Restaurants are:

1. Auntie Anne's (6002 Slide Road Space K108 (Kiosk)) (Ratings: 4.9)
2. Caprock Cafe (Ratings: 4.9)
3. Gong Cha 公茶 (405 Slide Rd, #119) (Ratings: 4.7)
4. Uncle Julio's Mexican (Lubbock) (Ratings: 4.7)
5. Raising Cane's (Ratings: 4.7)

Reply with a number from above to select the Restaurant

3. Restaurants by Price range [\$\$] ( \$ - Budget Friendly, \$\$ - Moderate, \$\$\$ - Expensive)

Response: List of restaurant for a given budget.

Example:

show Restaurants by Price range \$\$

According to the given range, these are the restaurants:

- 1 Caprock Cafe
- 2 Hook & Reel Cajun Seafood (6201 Slide Rd)
- 3 Hootie's Burger Bar (4950 South Loop 289)
- 4 Raising Cane's

Select a restaurant by entering it's serial number.

4. What are restaurants in [zipcode]

Response: List of restaurants in a given zipcode.

Example:

What are restaurants in 79416

1. Gong Cha 公茶 (405 Slide Rd, #119)

Reply with a number to select one of the above Restaurants.

5. Show me restaurants that are serving [Category]

Response: List of restaurants that serves a given category.

Example:

Show me restaurants that are serving pizza

Here is a list of restaurants serve pizza:

1. Earl of Sandwich (11824 Indiana Ave)
2. Panera (3004 W loop 289)

Select a restaurant by entering it's serial number.

6. Search for [Restaurant Name]

Response: Details of a given restaurants such as cuisine, price range, menu items, ratings.

Example:

search for Hook & Reel Cajun Seafood (6201 Slide Rd)

Please find restaurant Details

Name: Hook & Reel Cajun Seafood (6201 Slide Rd)

Ranking: 94

Rating: 0.2

Price Range: Expensive

Address: 6201 Slide Road, lubbock, TX, 79414

zipcode: 79414

Once you have picked a restaurant you can ask the following questions:

1. What cuisine this restaurant serves?

Response: List of cuisines the restaurant serves.

Example:

search for Hook & Reel Cajun Seafood (6201 Slide Rd)

Please find restaurant Details

Name: Hook & Reel Cajun Seafood (6201 Slide Rd)

Ranking: 94

Rating: 0.2

Price Range: Expensive

Address: 6201 Slide Road, Lubbock, TX, 79414

zipcode: 79414

What cuisine this restaurant serves?

Here's the Cuisine for 'Hook & Reel Cajun Seafood (6201 Slide Rd)':

1. American
2. Dinner
3. Seafood

2. Show me the menu.

Response: List of menu items along with their price.

Example:

whats the menu ?

1. Bahama Mama \$ 4
2. Gummy Worm \$ 4
3. Wonderwoman \$ 5
4. Guardians \$ 5
5. Hot Chai Latte \$ 5
6. Tuna Sub (Footlong) \$ 8
7. Lemon Ai Yu with White Pearl æ³æ³→â´â¼©æ„>çŽ‰ \$ 5
8. Milk foam Dirty Brown Sugar Milk Tea with Pearl \$ 6
9. Jalapeno Pretzel \$ 4
10. Pepperoni Pretzel Nuggets \$ 7
11. Cinnamon Sugar Pretzel Nuggets \$ 6
12. Original Pretzel Nuggets \$ 6
13. Mini Pretzel Dogs \$ 8
14. Glazed Donut Hole \$ 3
15. Jumbo Sausage with Cheese Kolache \$ 4

Type here...

send



3. What is budget range of this restaurant?

Response: Budget range of the restaurant like Moderate, Expensive.

Example:

what is the budget range for this restaurant ?

Expensive

4. Where is this restaurant located?

Response: Full address of the restaurant.

Example:

Where is this restaurant located?

Hook & Reel Cajun Seafood (6201 Slide Rd) is located at: 6201 Slide Road, lubbock, TX, 79414.



5. What are food categories served in this restaurant?

Response: List of food categories offered by the restaurant.

Example:

What are food categories served in this restaurant?

Here's the Food Categories for 'Hook & Reel Cajun Seafood (6201 Slide Rd)':

1. Picked for you
2. Appetizers
3. Soup and Salad
4. Combo Boils
5. Seafood Boils Small Catch
6. Seafood Boils Large Catch
7. Mini Boils
8. Po' Boys Sandwich
9. Baskets
10. Pasta
11. Sides
12. Drinks

# Modeling

---

## Objects

Id: unique Id for each restaurant.  
Position: Ranking of a restaurant.  
RestaurantName: Name of the restaurant.  
Score: number of people who rated the restaurant.  
Rating: rating of each restaurant.  
PriceRange: price range of each restaurant.  
Category: category of cuisine the restaurant serves.  
FullAddress: full address of the restaurant  
Zipcode: zip code of the restaurant.  
FoodCategory: food category that the restaurant serves.  
FoodDescription: description of each food item.  
FoodItem: name of the food item.  
foodPrice: price of the food item.

## Relations

name(X, Y): name of restaurant with id X is Y.  
position(X, Y): Y is the position of restaurant with id X.  
score(X, Y): Y is the score of the restaurant with id X.  
hasScore(X): denotes that restaurant with id X has a score.  
getScore(X, Y): denotes that restaurant with id X has a score Y.  
ratings(X, Y): denotes that restaurant with id X has a rating of Y.  
hasRating(X): denotes that restaurant with id X has a rating.  
getRating(X, Y): denotes that restaurant with id X has a rating of Y.  
category(X, Y): denotes that restaurant with id X has a category Y.  
priceRange(X, Y): denotes that restaurant with id X has a price range Y.  
hasPriceRange(X): denotes that restaurant with id X has a price range.  
getPriceRange(X, Y): denotes that restaurant with id X has a price range Y.  
fullAddress(X, Y): denotes that restaurant with id X has address Y.  
zipcode(X, Y): denotes that restaurant with id X has a zip code Y.  
positionByName(X, Y): denotes that restaurant with name X has position Y.  
scoreByName(X, Y): denotes that restaurant with name X has score Y.  
ratingsByName(X, Y): denotes that restaurant with name X has rating Y.  
categoryByName(X, Y): denotes that restaurant with name X has category Y.  
priceRangeByName(X, Y): denotes that restaurant with name X has price range Y.  
fullAddressByName(X, Y): denotes that restaurant with name X has address Y.  
zipcodeByName(X, Y): denotes that restaurant with name X has address Y.  
  
hasFoodCategory(#id, Y): denotes that restaurant with id X has food category Y.  
categoryHasItem(X, Y): denotes that food category X has food item Y.  
hasItem(X, Y): denotes that restaurant with id X has food item Y.

`price(X, Y)`: denotes that food item X has price Y.  
`description(X, Y)`: denotes that food item X has a description Y.  
`getRestaurantDetails(X, Y, P, S, R, P, A, Z)`: denotes that restaurant with id X has name Y, position P, score S, rating R, price range P, full address A and zip code Z.  
`getMenuByRestaurant(X, C, I, P)`: denotes that restaurant with id X has food categories C, food item I and food price P.  
`getRestaurantByCategory(C, X, Y)`: denotes that category C has restaurants with id X, and name Y.  
`zipcodeByNameAndId(X, Y, Z)`: denotes that restaurant with id X has name Y and zip code Z.

## Knowledge/rules

% Restaurant id has name, position, score, ratings, price range, full address and zip code.  
`getRestaurantDetails(I, N, P, S, R, B, A, Z) :- name(I, N), position(I, P), score(I, S), ratings(I, R), priceRange(I, B), fullAddress(I, A), zipcode(I, Z).`

% Restaurant id has food categories, items, and price food each item  
`getMenuByRestaurant(I, C, F, P) :- hasFoodCategory(I, C), categoryHasItem(C, F), price(F, P).`

% Category has restaurants  
`getRestaurantByCategory(C, I, R) :- category(I, C), name(I, R).`

% Restaurant has a price range  
`priceRangeByName(X, Y) :- name(I, X), priceRange(I, Y).`

% Restaurant has a zip code  
`zipcodeByNameAndId(I, N, Z) :- zipcode(I, Z), name(I, N).`

% default price range for a restaurant is 2.  
`hasPriceRange(X) :- priceRange(X, Y).`  
`getPriceRange(X, Y) :- priceRange(X, Y).`  
`getPriceRange(X, 2) :- not hasPriceRange(X).`

% Closed world assumption for Price Range  
`-getPriceRange(X, Y) :- not getPriceRange(X, Y).`

% default rating for a restaurant is 0 if it does not have a rating.  
`hasRating(X) :- ratings(X, Y).`  
`getRating(X, Y) :- ratings(X, Y).`  
`getRating(X, 0) :- not hasRating(X).`

% default score of a restaurant is 25 if it does not have a score.  
`hasScore(X) :- score(X, Y).`  
`getScore(X, Y) :- score(X, Y).`  
`getScore(X, 25) :- not hasScore(X).`

# Web Application Interface Design

The Web Application is interfaced through a chat dialogue which takes input in the form of both text and Voice and the application responds with both text and voice as described below:

The Web Application is interfaced through a chat dialogue which takes input in the form of both text and Voice and the application responds with both text and voice as described below:

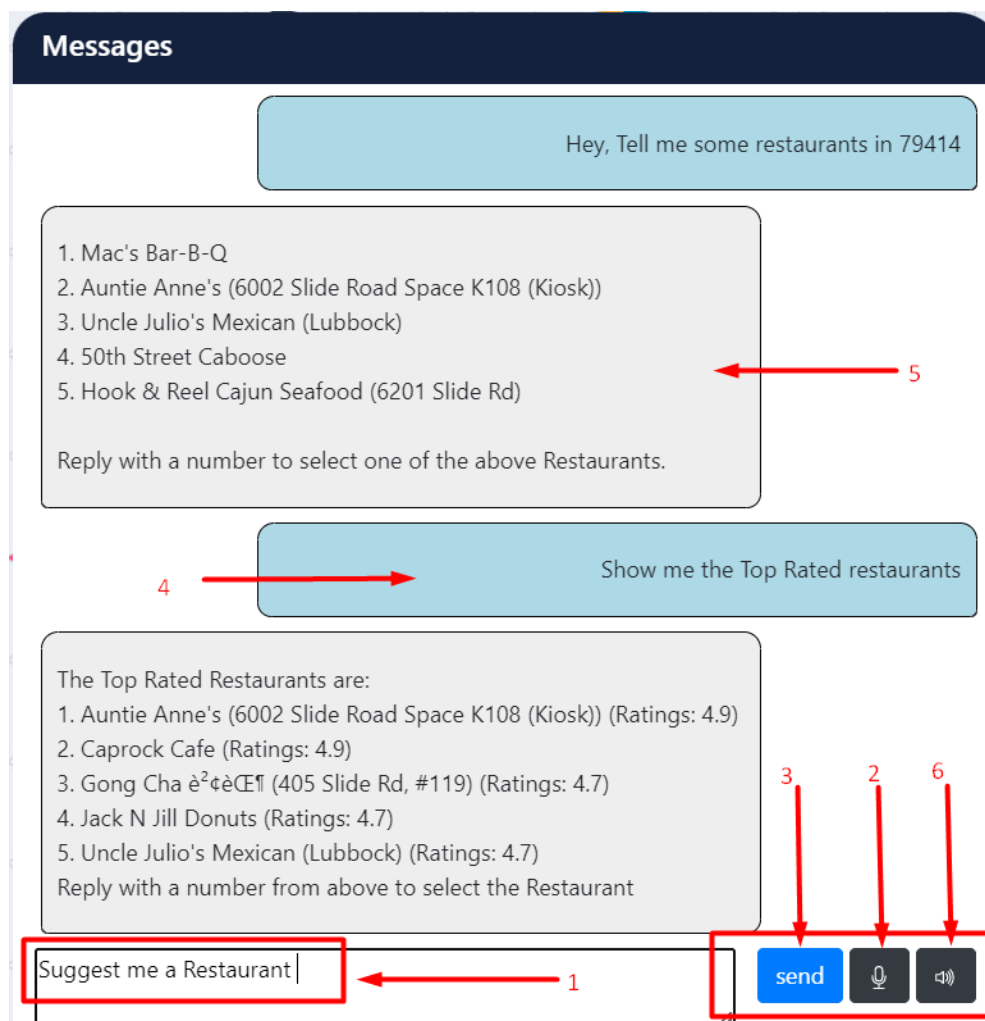


Figure 1.1

The Application is a Single Webpage Application where all the interactions Happen using a chat dialogue as shown in the above Figure 1.1.

## Input

The User can type a query in the text area as shown by the arrow 1 in the figure 1.1 or the user can click on the Mic button which is the arrow 2 option and Speak the query in English using the computers microphone then the speech will be synthesized to text and the text will be shown in the text area(arrow 1) so that the text can be for further corrected if required, and the input can be send using the send button which is the option highlighted using the

arrow 3, using which a query can be sent to the application and thus the user can send input to the application using either text or speech.

All queries are shown in the chat box dialogue as highlighted by arrow 4 in the Figure 1.1.

## Output

The response from the application will be shown in chat dialogue after the input, like an example the response for the text *“Hey, Tell me some restaurants in 79414”* is highlighted above using arrow 5. By default, all responses in text will be read out in English which is controlled by the volume toggle option highlighted by arrow 6, using which the user can enable or disable the voice output.

# Implementation of Web Application

This application is a single page web application which uses, Bootstrap for styling the webpage along with some custom CSS, jQuery to provided fast and easy dynamic behaviour to our web page and to make Ajax asynchronous http requests, and other custom javascript files to control the overall behaviour of the chatbot and to introduce custom required functionalities that are needed for the Restaurant suggestion chatbot. The Web application also uses Web Speech API provided by the browsers to convert 'text to speech' and 'speech to text' and the Application makes requests to CORS-ANYWHERE to bypass cross origin request error's and uses wave.ttu.edu/ajax.php to execute queries for a given knowledge and to get the required answer sets to the web applications.

The below diagram depicts the components of the whole system and the high-level overview of the entire application.

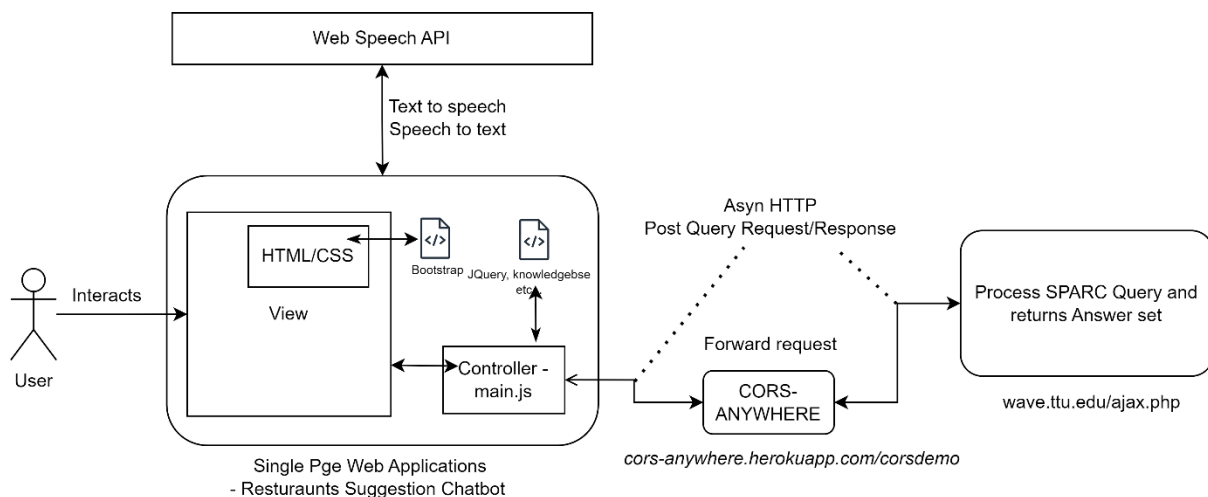


Figure 2: High level component Diagram.

## Further Details on the above components:

- **The View:**

The View is a simple HTTP webpage which uses Bootstrap and style.css for adding styles to the user interface. It has a simple repeatable background and has required tags such as id and class for all the required DOM elements which can be used to in select query for DOM manipulations or adding/changing styles as and when required.

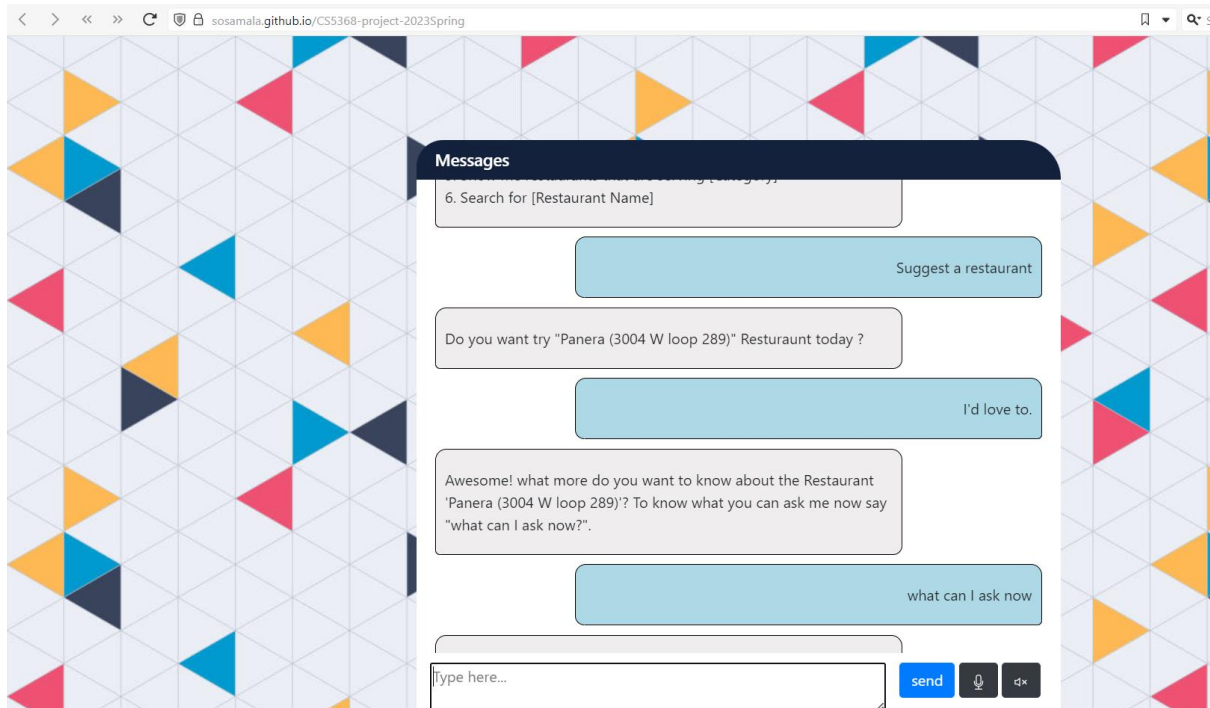


Figure 3: The View/ User Interface.

### • The Controller/Logic.

The controller which is the logic in the main.js file is the main file, which uses different libraries such as jQuery, Web Speech API and make external asynchronous http requests to fetch answer sets using the [wave.ttu.edu/ajax.php](http://wave.ttu.edu/ajax.php) , by making a POST request via the cors-anywhere application. Along with it is also responsible for Make DOM changes such as representing the text messages and also providing the functionalities for the Text-Speech and Speech-text synthesis. It is also responsible for text processing and extraction of required features from the text and converting them to queries which can be processed by external application using the data present in the knowledgebase.js file (The SPARC program), to get the answer set for the required program.

#### *Functionalities:*

##### 1. Context Object:

The Context Object is a special object which details the current context of the conversation. It shared between all the conversations/messages and is updated to have the current snapshot of the crux of the conversation so that the response can be accurately provided. We have identified the below fields that are required to process a conversation for this application as follows:

1. Restaurant id
2. Restaurant Name
3. Expecting a response
4. Saved results

Saved results hold the results from the previous relevant conversation results such as the top 5 restaurants that are showed to the user, or the suggested restaurant, or the list of restaurants serving a dish in their menu.

Expecting a response is a 'tag' that allows the web application to process the request of user as a response to a previous query from the chatbot from the past conversation or a new request. Tags are discussed further in detail below.

## 2. Tags:

A tag in the web application is used to identify a process in the web application, Different processes are used to perform different tasks and a tag is associated with each task, A task can branch into multiple ways, Such as after selecting a restaurant, the user can search for another restaurant or select a dig into the restaurant as ask for its location or menu etc. Therefore, this could happen over multiple messages, therefore, to know which task to perform at any given moment we use a tag and switch over its values to perform a task using the tag and additional details in the context object. The tags are referred to as "response" in bot-commands.js file.

## 3. Text Pre Processing Functionality:

The text pre-Processing functionality is used to process the message from the user and convert it to a query which can be answered by using the domain knowledge(sorts, predicate, rules) and a SPARC solver. The web application uses different levels of sophistication to map a user's request/message to task (Tag) which are mentioned below.

### a. The Percentile Method:

The Percentile Method uses collection of lists of possible words and another lists of must\_have words, each associated with a tag. If the user message contains all the tokens from the must\_have words list then it calculates the percentage of hits for the words in the possible words list. The list of words which has the the highest amount of hits will be chosen as the corresponding tag will be processed and the required tasks will be performed.

Sudo Code:

```
percentile_collection = [
    [
        words = [...],
        must_have = [...],
        tag = [...]
    ],
    [
        words = [...],
        must_have = [...],
        tag = [...]
    ],
    ...
]

def process_percentile(userMessage) {
    tokens = userMessage.split(" ")
    best_hit_index = -1
    best_hits = -1

    # 1. for each value in the percentile_collection
    # 2. check if tokens has all the must_have words
```



```

# 3. calculate the hits
# 4. check if the current_hits > best_hits if so
#   Update the best_hit_index
# return if any hits otherwise null
return {data: percentile_collection[best_hit_index]}
}

```

This method is best used if there are multiple ways to convey the same message using short number of words, and does not have any inputs that needs to be extracted.

#### b. The Regex Method:

The Regex method uses the same pattern as above and has a collection of lists with an array of regex patterns along with the tasks to identify the correct tag and perform the right tasks. This method is best used for long sentences and also to extract words/features/inputs from the user as they can be well defined in a pattern.

Sudo Code:

```

percentile_regex = [
  [
    regex_pattern = [...],
    tag = [...]
  ],
  [
    regex_pattern = [...],
    tag = [...]
  ],
  ...
]

def process_regex(userMessage) {

  # 1. for each value in the percentile_regex, say i
  # 2. check if userMessage matches with any of the pattern in
  regex_pattern
  for(p in regex_pattern) {
    reg = RegExp(p);
    response = reg.exec(userMessage);
    if(response != null) return { "regex" : response , "data":
percentile_regex[i]};
  }
  # 3. else return null
  return null
}

```

Here we can use the 'response' Object to extract specific input/feature/information from users input such as the restaurant name, the selection budget [\$,\$\$,,\$\$,\$\$] etc.

The Above configurations for Text Proccessioning could be found in bot-commands.js file.

Now once, we have an understanding on how a message can be processed and assigned to a task/tag and extract the required features from the text, all there is left to do is to do is call process the inputs from the users in the below order so that we can facilitate the chat functionality.

Step 1: clean the message, remove unrequired characters from the message which are not needed.

```
# clean the message
message = message.toLowerCase().trim().replace(/[^\a-z0-9$ ]/g, "");
```

Step 2: Check if the message is a response to another message or a new request using the context object.

```
#process if expecting a response first
if(context != null && context.expecting_a_response != null) {
    var ex_return = process_exc(message);
    if(ex_return != null) {
        return process_msg_response(process_exc_response(ex_return));
    }
}
```

Step 3: if not step 2, check if the message matches any of the regex strings as its computationally faster to do so.

```
#process regex second
var pr_return = process_regex(message);
if(pr_return != null) {
    return
process_msg_response(process_regex_response(pr_return));
}
```

Step 4: if not step 3, check if the message matches any of the percentile patterns in the last.

```
#process percentile last
var pp_return = process_percentile(message);
if(pp_return != null) {
    return
process_msg_response(process_percentile_response(pp_return));
}
```

Step 5: if not step 4, return default 'unable to understand' message.

```
#return default message
process_msg_response({"data": "I don't understand what you said, could you please retry? Thanks :)"});
```

Thus, a Text is pre-processed and the required features are extracted to create the required query and send to external system for answer sets.

#### 4. Making Ajax Requests and getting the answer sets:

Using the extracted feature the Context object and the task tag, we can easily identify the query that is required to be run using a switch case. Using the \$.ajax method we can make the required call the external system to get the answer sets.

Sudo Code, Main Method:

```
$.ajax({
  url: "https://cors-
anywhere.herokuapp.com/http://wave.ttu.edu/ajax.php",
  type: "POST",
  headers: {
    "X-Requested-With": "XMLHttpRequest",
    "Access-Control-Allow-Origin": "*",
    "Access-Control-Allow-Methods" : "GET, POST, PUT, PATCH, DELETE,
OPTIONS"
  },
  data: {
    action: "getQuery",
    query: query,
    editor: knowledge
  },
  success: function(response ) {
    process_msg_response(processAjaxResponse(response, unknowns,
pr_return ));
  },
  error: function(xhr, status, error) {
    console.log("error: " + error);
  }
});
```

In the above function call, the query string is formed used the relationship name which corresponds to the task/tag, context object. The inputs from the users along with some unknowns in the required places.

Sudo code:

```
def process_regex_response(pr_return) {
  ...
  input_extracted_features = [...]
  if(TAG === 'q1'){
    params = []
    unknowns = 0;
    params.push(unknown_names[unknowns++]) #unkown value
    params.push(input_extracted_features[0]) # known value
    params.push(unknown_names[unknowns++]) #unkown value
    query_name = 'name'; # Relationship name for the TAG
    return createAndMakeQuery(query_name, params, unknowns,
pr_return); #query = name(X,knownValue, Y)
  }
  ...
}
```

Knowledge is the entire SPARC Program on which a query will be runed, the SPARC program for this application is stored in a variable knowledge which could be found in 'knowledgeBase.js'

The response can be easily parsed using a regex pattern matcher after splitting it first based '<P' on '<br' and looking for a pattern such as 'unkown\_values[i] = (.\*)'

Sudo code:

```
#For each unknown value i
  rStr = '(.*)' + unknown_names[i] + ' = ([0-9a-zA-Z]*) (.*)';
  answer = RegExp(rStr).exec(line+ " filler");
```

### 5. Creating the response message:

Once the response is parsed from the external system and the unknown values are extracted again using the context object, the task(tag) value we can put the extracted information into predefined meaningful text using some string manipulation techniques.

Sudo Code:

```
if(TAG === "q1") {

  response_message = '';
  ...
  context.expecting_a_response = "q1";
  context.saved_suff = {
    "data" : item
  };
  if(!neg_response)
    response_message = 'Do you want try "{}" Resturaunt today
?'.format([res_name]);
}
```

### 6. Input: Speech to text:

We use the Web Speech API to convert the voice input from user to text using the predefined methods such as `recognition.start` and `recognition.onresult` methods, where `recognition` is a new `SpeechRecognition()` object. The `start` method is called when the user clicks on the mic button and the `onresult` method is called the Web Speech API with the required text converted to speech.

Sudo Code:

```
/* Speech Recognition */
const SpeechRecognition = window.SpeechRecognition ||
webkitSpeechRecognition;
const SpeechGrammarList = window.SpeechGrammarList ||
webkitSpeechGrammarList;
const SpeechRecognitionEvent = window.SpeechRecognitionEvent ||
webkitSpeechRecognitionEvent;
const recognition = new SpeechRecognition();
const speechRecognitionList = new SpeechGrammarList();
recognition.continuous = false;
recognition.lang = "en-US";
recognition.interimResults = false;
recognition.maxAlternatives = 1;

$("#btMic").click(function(e) {
  recognition.start();
  console.log("Listening...");
});

recognition.onresult = (event) => {
  text = event.results[0][0].transcript
  # Updat the UI with the text
};
```

## 7. Output: Text to speech:

We use the Web Speech API to Convert the required text to speech, using the methods provided such as `synth.speak(textToUtter)` and `synth.cancel()` to Utter speech or cancel when the user disables the toggle. Here 'textToUtter' is an `SpeechSynthesisUtterance` instance.

Sudo Code:

```
/* Speech Synthesis*/
const synth = window.speechSynthesis;
const speech_voice_name = synth.getVoices()[10];
const speech_rate = 1.0
const specch_pitch = 1.0

def recieveMessage(message) {

    # Update UI

    if(isVolume) {
        synth.cancel();
        const utterThis = new SpeechSynthesisUtterance(message);
        utterThis.voice = synth.getVoices()[2];
        utterThis.pitch = specch_pitch;
        utterThis.rate = speech_rate;
        # Utter the text
    }
}
```

# Project Structure

---

Here's an overview of the files and folders in the project folder and brief purpose:

```
CS5368-project-2023Spring:
|
|   favicon.ico # Favuoirte Icon
|   index.html  #The Main view of the web page
|   mid-report.docx #Mid Report
|   mid-report.pdf #Mid Report
|   final-report.docx #Final Report
|   final-report.pdf #Final Report
|   README.md #Read me for the project
|
|   └── css
|       |
|       |   bootstrap-grid.min.css #Bootstrap Grid CSS library file
|       |   bootstrap-grid.min.css.map #Bootstrap Grid CSS library map file
|       |   bootstrap-reboot.min.css #Bootstrap Reboot CSS library file
|       |   bootstrap-reboot.min.css.map #Bootstrap Reboot CSS library map file
|       |   bootstrap.min.css #Bootstrap CSS library file
|       |   bootstrap.min.css.map #Bootstrap CSS library map file
|       |   bootstrap.rtl.min.css.map #Bootstrap CSS library rtl map file
|       |   style.css # Custom CSS styles for the project.
|       |
|       |   └── fonts
|           |
|           |   bootstrap-icons.css #bootstrap icon library css file
|           |   bootstrap-icons.json #bootstrap icons library json files
|           |   fonts.zip #A bunch of fonts zipped.
|           |   mic.svg #mic icon svg
|           |   volume-mute.svg #volume mute icon
|           |   volume-up.svg #volume up icon
|           |
|           |   └── fonts
|               |
|               |   bootstrap-icons.woff #bootstrap icon fonts
|               |   bootstrap-icons.woff2 #bootstrap icons fonts
|               |
|               |   └── datsource
|                   |
|                   |   datamaker.py #Python file, add hashes to CSv Data
|                   |   food_new.db # SQLite Food Database
|                   |   Queries_V1.sql #Queries SQL
|                   |   res_cat_lubbock_main.csv #Lubbock Restaurant Categories
|                   |   res_lubbock_main.csv #Lubbock Restaurants
|                   |   res_menus_lubbock_main.csv #Lubbock Restaurants Menu
|                   |
|                   |   └── img
|                       |
|                       |   check2.jpg #Background Image
|                       |   food_tile_1.jpg #background Image2
|                       |
|                       |   └── js
|                           |
|                           |   bootstrap.min.js #bootstrap library minify JS
|                           |   bot-commands.js #JS File for Bot Commands logic
|                           |   constants.js #JS File to hold Constatns
|                           |   data.js #JS File to hold Hashes to Data and vice versa mapping
|                           |   jquery-3.6.4.min.js # JQuery Library
|                           |   knowledgeBase.js # SPARC program
|                           |   main.js # Main Logic for Web page Interactions.
```

# Techniques

---

## Web Technologies:

- **HTML**

Used HTML to create webpages for the Restaurant intelligent agent.

- **JavaScript (jQuery, Ajax)**

Used JavaScript and jQuery framework to create logic and to add various controls for the web site as listed below.

1. Process text/speech from user and translate it to the required SPARC query and respond with text/speech.

We have used the web speech API<sup>[1]</sup> to synthesize the responses of the user and also get the text input from user from voice commands. Then we have developed a multi-layer text processing function which first processes the input from user based on the context and expected response, if that fails then checks for the Regex Pattern for various questions and checks if the input matches any of those and processes the request accordingly, if that also fails then we check the input against a collection of strings to check the best word count match using percentage of hits to determine if the input matches any of those, if that also fails then we respond with a fall back text.

Once we get a regex, or pattern matched in the above process, in some cases we may make a query request to SPARC to get results for which we have used Ajax (Step 3) web requests and once we get the corresponding response from Sparc we again used regex to process the corresponding variables/unknowns, extract the required data from the parameter query results and translate the extracted data into a sentence to respond the user.

2. Various User interface actions such as keyboard and button click actions have been mapped to override defaults and perform a certain action, like enter key on text input send the message, the toggle for voice input/output etc.
3. We have used Ajax (Asynchronous JavaScript and XML) web technology to create asynchronous web applications and requests to pass the Sorts, Predicates, Knowledge/Rules along with the required query to <http://wave.ttu.edu/ajax.php> via the <https://cors-anywhere.herokuapp.com/> to overcome the CORS(Cross-origin resource sharing) issue when accessing the ASP SPARC features from wave.ttu.edu through github.io domain. However, to enable this, one must request temporary access to the demo server at <https://cors-anywhere.herokuapp.com/corsdemo>.

- **CSS(BOOTSTRAP)**

We have used Bootstrap and other custom CSS style sheets to make the User Interface more appealing to the user.

## ASP: SPARC

- Data source extraction Hashing required fields using Python

We have used a python script to generate unique Hash values for various fields of the Restaurant Data Source so that they can be used as value names for the 'sorts' in ASP program which are unique strings mapping to the real-world String data stored in the JavaScript maps. Since many values start with numeric or capital letters or have special characters etc, in them the hash is used instead to establish the relationships and other rules for different sorts whereas the JavaScript map has the actual values. The hash is an alphabet unique string code which can be used as names for different sorts.

- Generating relationships using SQLite DB queries

We have used the restaurant data source values in SQLite database and wrote queries to quickly generate the required SPARC sort definition to data mappings between objects. We have normalized/or rearranged the data where applicable in the restaurant data source to improve the performance of the DB queries and quickly generate the SPARC relationships which are straight forward.

- Default Values

We have written Sparc rules to generate default values wherever necessary and writer other rules to enhance the object relationships in SPARC.

## GitHub

We have hosted the required source code and data source to GitHub repository and hosted the web application in GitHub<sup>[3]</sup>.



## References

---

1. *Using the web speech API - web apis: MDN*. Web APIs | MDN. (n.d.). Retrieved April 1, 2023, from [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Speech\\_API/Using\\_the\\_Web\\_Speech\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API/Using_the_Web_Speech_API)
2. Liyuan-Gao. (n.d.). *Liyuan-Gao/asp\_example: A simple example of web application for ASP program*. GitHub. Retrieved April 1, 2023, from [https://github.com/liyuan-gao/ASP\\_example](https://github.com/liyuan-gao/ASP_example)
3. “Creating a GitHub Pages site,” *GitHub Docs*. <https://docs.github.com/en/pages/getting-started-with-github-pages/creating-a-github-pages-site> (accessed Apr. 02, 2023).