## KEY BOT COMPONENTS

- Ability to look up and send customer documents (Oscar/Siebel)

- Ability to answer apartment costing/pricing questions (BI or Oscar/Siebel)

- Answer Tenancy related questions (Oscar/Siebel)

- Answer Availability questions (eg. Leverage existing API’s) (BI warehouse w/ Kaleb)

- Lookup Service Track tickets and inquire on status (Alex/Gary)

- Ask for property phone numbers (Directory of active phone numbers?)

- Ask for local attractions near property (Google Places API?) Yelp API

- Average price hardcoded from BI data

- Frequently asked questions

- Analyze google spreadsheet data

- Get weather information

- Create helpdesk tickets

- Request a room to reserve

- Can email data entered back to company

- Can store data into the google cloud

- Integrates with Slack, Facebook Messenger, Cisco, Webpage, Twitter, Skype, Alexa, Cortana, Kik, Assistant

**Objective**

Prototype conversational dialog using one of the preferred technologies and develop vision for how Oakwood can leverage AI/ML for differentiation with it’s clients and guests.

**Initial focus**

Chat bot to support Customer Service at WSSC with customer requests.

**Longer term**

This can also be consumer facing.

**Preferred Technology**

Google Dialog Flow, IBM Watson or Amazon  
See attached Forrester Report and determine best approach for information on Conversational Computing Platforms.  
  
**Additional Resources**  
- *Oscar Team*(Alex and Joanna) – can provide guidance on information from existing API  
- *Invoice and Payment docs*(Joanna) – available thru Paylease and Oscar  
- *Service Track Tickets/stats*(Alex and Gary)   
- *Siebel*(Shankar) – can provide guidance on Hippo content integration and Siebel pricing API’s  
- *BI Pricing Widget*(Kaleb/Stacy) – can provide guidance on apartment costs by unit type and city

## OTHER PROJECTS

## - Recreating ACE Survey and ServiceTrack Survey in Survey Monkey to interface for FB Messenger

## - Automated tool to test regression testing on mobile app (work with Photon)

## - Re-imagine support.oakwood.com

## - Develop Security Training on SharePoint

## - Annual Inventory for Field

## - Remove unused devices at Corporate

## - Review and organize SharePoint

## TIMELINE

**31st-7th:**

* ~~Transfer bot to kevins new google enterprise account~~

**7th-14th:**

* ~~Get connected to live pricing data using our flask server~~
* ~~fine tune pricing function (maybe add state/country/long/lat key codes to return similar locations if query doesn’t return anything)~~

**14th-21st:**

* Train unknown questions (Ex: Who is bill keffler?, phone number directory)
* Fine tune/train completed questions

**21st-28th:**

* Decide how we want to split apart bots. Where does the pricing bot go? Where does the FAQ bot go? (Ex: connect to fb page, put on website, put in app, phone number)

**28th-4th:**

* Side projects? Create template for survey, support.oakwood, sharepoint
* Present to IT managers

**4th-11th:**

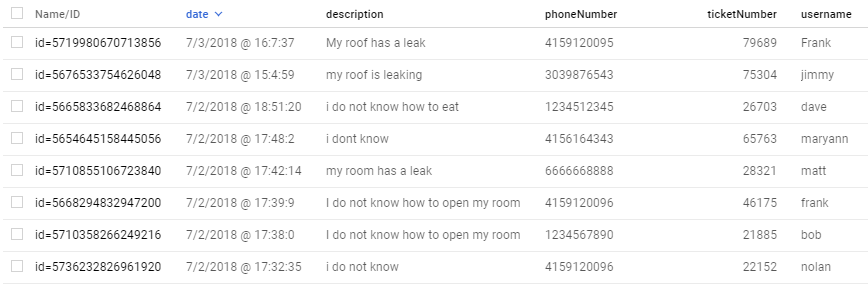
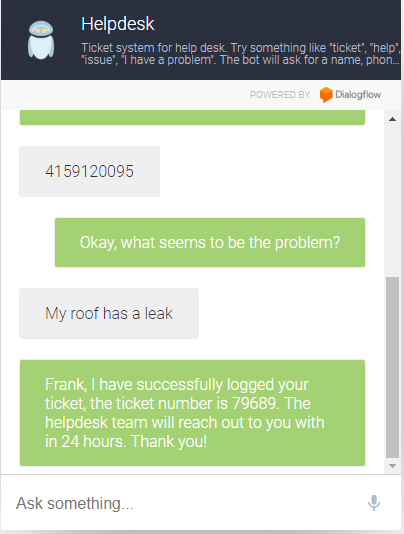
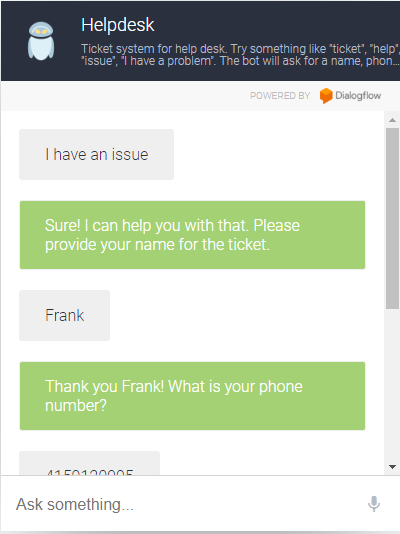
* Make and finalizations to bots
* Final presentations on 10th

## BOTS

### HELPDESK BOT

<https://bot.dialogflow.com/OakwoodHelpdesk>

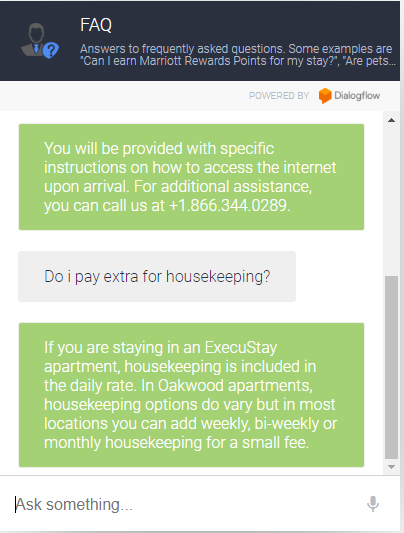
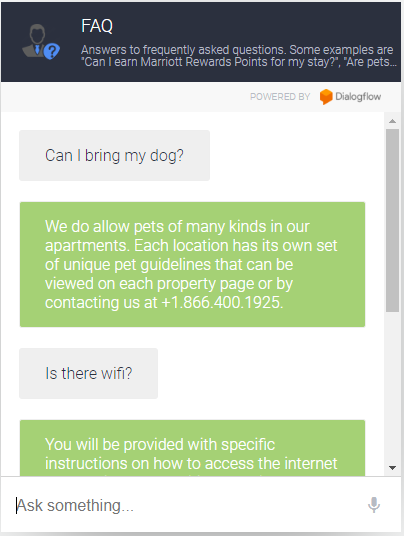
Ticket system for help desk. Try something like "ticket", "help", "issue", "I have a problem". The bot will ask for a name, phone number, and description of the problem. A random ticket number will be assigned to the issue. The data will be stored into a Google Cloud ticket query for later access.



### FAQ BOT

<https://bot.dialogflow.com/OakwoodFAQ>

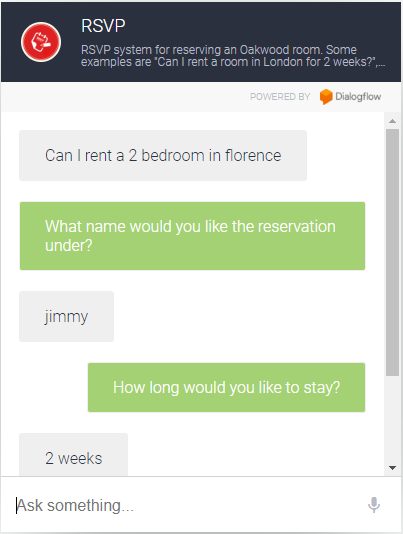
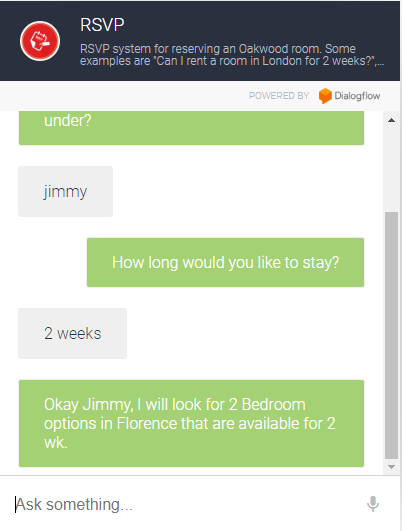
Answers to frequently asked questions. Some examples are "Can I earn Marriott Rewards Points for my stay?", "Are pets allowed in your apartments?", "Is housekeeping included?", "How to log in to the Wi-Fi in the apartment?".

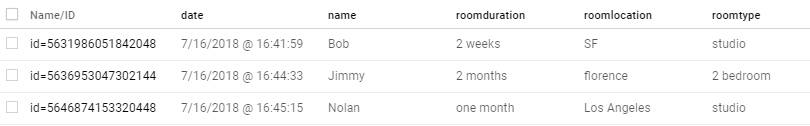


### RSVP BOT

<https://bot.dialogflow.com/OakwoodRSVP>

RSVP system for reserving an Oakwood room. Some examples are "Can I rent a room in London for 2 weeks?", "Can I reserve a single?", "Can I rent?", "I want to rent a studio for one month.", "I want an apartment in SF.". All data is saved an pushed to the cloud.

****

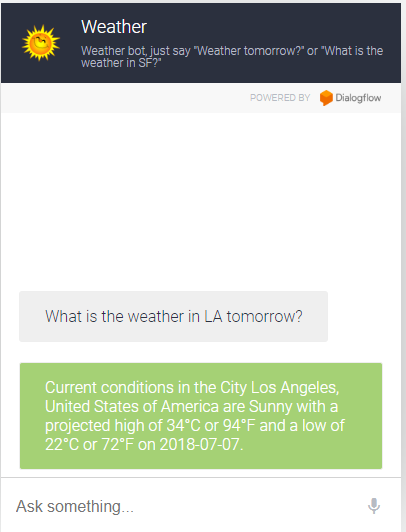
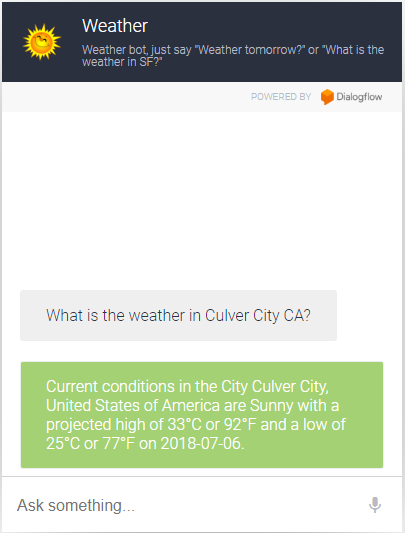


### 

### WEATHER BOT

<https://bot.dialogflow.com/OakwoodWeather>

Weather bot, just say "Weather tomorrow?" or "What is the weather in SF?"



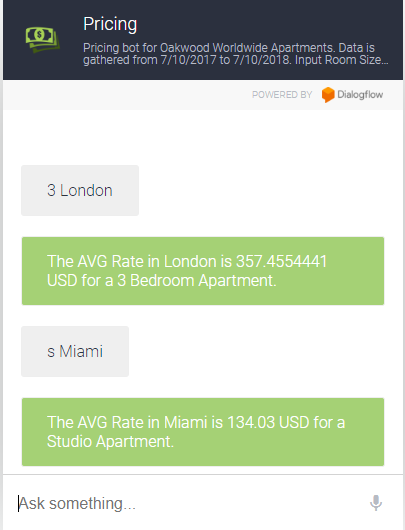
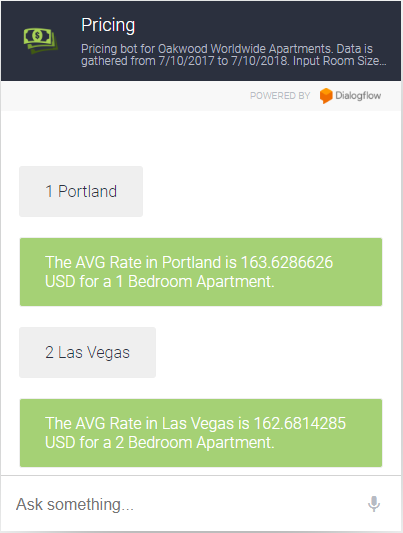
## 

### PRICING BOT

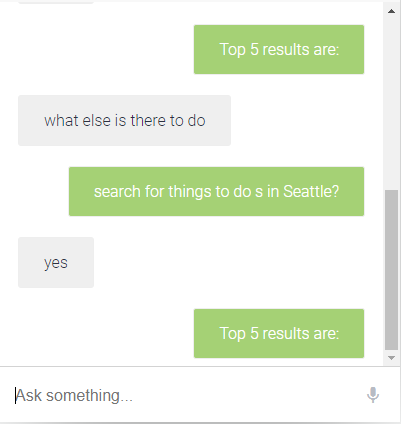
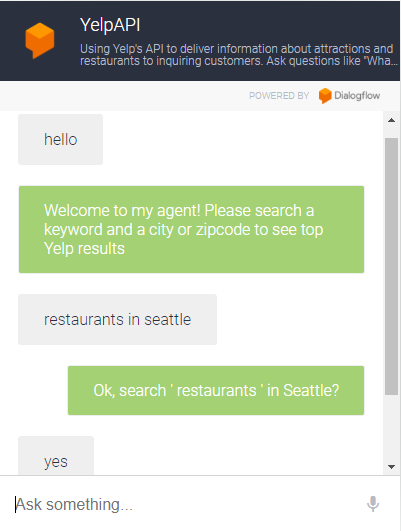
<https://bot.dialogflow.com/OakwoodPricing>

(hardcoded w/ csv files from BI db)

Pricing bot for Oakwood Worldwide Apartments. Data is gathered from 7/10/2017 to 7/10/2018. Input Room Size and Location. For example try "1 Portland", "Double in Las Vegas", "3 London", "Studio Miami" to get started.



### YELP SEARCH BOT



### 

### POPULAR LOCATION FAQ BOT

### - some cities and states typed in as entities. Don’t really want to hardcode more - hook up to DB?

### 

## BOT FULFILMENTS

### HELPDESK FULFILLMENT

'use strict';

const http = require('http');

// Imports the Google Cloud client library

const Datastore = require('@google-cloud/datastore');

// Your Google Cloud Platform project ID

const projectId = 'helpdesk-104b6';

// Instantiates a client

const datastore = Datastore({

projectId: projectId

});

// The kind for the new entity

const kind = 'ticket';

const {dialogflow} = require('actions-on-google');

const functions = require('firebase-functions');

const app = dialogflow({debug: true});

function listTasks() {

const query = datastore.createQuery('ticket').order('date');

datastore

.runQuery(query)

.then(results => {

const tasks = results[0];

console.log('Tasks:');

tasks.forEach(task => {

const taskKey = task[datastore.KEY];

console.log(taskKey.id, task);

});

})

.catch(err => {

console.error('ERROR:', err);

});

}

exports.dialogflowFirebaseFulfillment = (req, res) => {

console.log('Dialogflow Request body: ' + JSON.stringify(req.body));

// Get the city and date from the request

let ticketDescription = req.body.queryResult['queryText']; // incidence is a required param queryText

//let name = req.body.result.contexts[0].parameters['given-name.original'];

let username = req.body.queryResult.outputContexts[1].parameters['given-name.original'];

let phone\_number = req.body.queryResult.outputContexts[1].parameters['phone-number.original'];

let currentdate = new Date();

let datetime = (currentdate.getMonth()+1) + "/"

+ currentdate.getDate() + "/"

+ currentdate.getFullYear() + " @ "

+ currentdate.getHours() + ":"

+ currentdate.getMinutes() + ":"

+ currentdate.getSeconds();

console.log('date is ' +datetime);

console.log('description is ' +ticketDescription);

console.log('name is '+ username);

console.log('phone number is '+ phone\_number);

function randomIntInc (low, high) {

return Math.floor(Math.random() \* (high - low + 1) + low);

}

let ticketnum = randomIntInc(11111,99999);

// The Cloud Datastore key for the new entity

const taskKey = datastore.key(kind);

// Prepares the new entity

const task = {

key: taskKey,

data: {

date: datetime,

description: ticketDescription,

username: username,

phoneNumber: phone\_number,

ticketNumber: ticketnum

}

};

console.log("incidence is " , task);

// Saves the entity

datastore.save(task)

.then(() => {

console.log(`Saved ${task.key}: ${task.data.description}`);

res.setHeader('Content-Type', 'application/json');

//Response to send to Dialogflog

res.send(JSON.stringify({ 'fulfillmentText': "I have successfully logged your ticket, the ticket number is " + ticketnum + ". Someone from the helpdesk will reach out to you within 24 hours. "}));

//res.send(JSON.stringify({ 'fulfillmentText': "I have successfully logged your ticket, the ticket number is " + ticketnum + ". Someone from the helpdesk will reach out to you within 24 hours.", 'fulfillmentMessages': "I have successfully logged your ticket, the ticket number is " + ticketnum + ". Someone from the helpdesk will reach out to you within 24 hours."}));

})

.catch((err) => {

console.error('ERROR:', err);

res.setHeader('Content-Type', 'application/json');

res.send(JSON.stringify({ 'speech': "Error occurred while saving, try again later", 'displayText': "Error occurred while saving, try again later" }));

});

};

### MIXED FULFILLMENT

'use strict';

const http = require('http');

// Imports the Google Cloud client library

const Datastore = require('@google-cloud/datastore');

// Your Google Cloud Platform project ID

const projectId = 'helpdesk-104b6';

// Instantiates a client

const datastore = Datastore({

projectId: projectId

});

// The kind for the new entity

const kind = 'ticket';

const {dialogflow} = require('actions-on-google');

const functions = require('firebase-functions');

const {WebhookClient} = require('dialogflow-fulfillment');

const {Card, Suggestion} = require('dialogflow-fulfillment');

process.env.DEBUG = 'dialogflow:debug'; // enables lib debugging statements

exports.dialogflowFirebaseFulfillment = functions.https.onRequest((req, res) => {

const agent = new WebhookClient({ req, res });

console.log('Dialogflow Request headers: ' + JSON.stringify(req.headers));

console.log('Dialogflow Request body: ' + JSON.stringify(req.body));

function submit(agent) {

let ticketDescription = agent.req.body.queryResult['queryText']; // incidence is a required param queryText

//let name = req.body.result.contexts[0].parameters['given-name.original'];

let username = agent.req.body.queryResult.outputContexts[1].parameters['given-name.original'];

let phone\_number = agent.req.body.queryResult.outputContexts[1].parameters['phone-number.original'];

let currentdate = new Date();

let datetime = (currentdate.getMonth()+1) + "/"

+ currentdate.getDate() + "/"

+ currentdate.getFullYear() + " @ "

+ currentdate.getHours() + ":"

+ currentdate.getMinutes() + ":"

+ currentdate.getSeconds();

console.log('date is ' +datetime);

console.log('description is ' +ticketDescription);

console.log('name is '+ username);

console.log('phone number is '+ phone\_number);

function randomIntInc (low, high) {

return Math.floor(Math.random() \* (high - low + 1) + low);

}

let ticketnum = randomIntInc(11111,99999);

// The Cloud Datastore key for the new entity

const taskKey = datastore.key(kind);

// Prepares the new entity

const task = {

key: taskKey,

data: {

date: datetime,

description: ticketDescription,

username: username,

phoneNumber: phone\_number,

ticketNumber: ticketnum

}

};

console.log("incidence is " , task);

// Saves the entity

datastore.save(task)

.then(() => {

console.log(`Saved ${task.key}: ${task.data.description}`);

agent.add("I have successfully logged your ticket, the ticket number is " + ticketnum + ". Someone from the helpdesk will reach out to you within 24 hours. ");

//res.send(JSON.stringify({ 'fulfillmentText': "I have successfully logged your ticket, the ticket number is " + ticketnum + ". Someone from the helpdesk will reach out to you within 24 hours.", 'fulfillmentMessages': "I have successfully logged your ticket, the ticket number is " + ticketnum + ". Someone from the helpdesk will reach out to you within 24 hours."}));

})

.catch((err) => {

console.error('ERROR:', err);

res.setHeader('Content-Type', 'application/json');

});

}

// // Uncomment and edit to make your own intent handler

// // uncomment `intentMap.set('your intent name here', yourFunctionHandler);`

// // below to get this function to be run when a Dialogflow intent is matched

// function yourFunctionHandler(agent) {

// agent.add(`This message is from Dialogflow's Cloud Functions for Firebase editor!`);

// agent.add(new Card({

// title: `Title: this is a card title`,

// imageUrl: 'https://developers.google.com/actions/images/badges/XPM\_BADGING\_GoogleAssistant\_VER.png',

// text: `This is the body text of a card. You can even use line\n breaks and emoji! 💁`,

// buttonText: 'This is a button',

// buttonUrl: 'https://assistant.google.com/'

// })

// );

// agent.add(new Suggestion(`Quick Reply`));

// agent.add(new Suggestion(`Suggestion`));

// agent.setContext({ name: 'weather', lifespan: 2, parameters: { city: 'Rome' }});

// }

// // Uncomment and edit to make your own Google Assistant intent handler

// // uncomment `intentMap.set('your intent name here', googleAssistantHandler);`

// // below to get this function to be run when a Dialogflow intent is matched

// function googleAssistantHandler(agent) {

// let conv = agent.conv(); // Get Actions on Google library conv instance

// conv.ask('Hello from the Actions on Google client library!') // Use Actions on Google library

// agent.add(conv); // Add Actions on Google library responses to your agent's response

// }

// // See https://github.com/dialogflow/dialogflow-fulfillment-nodejs/tree/master/samples/actions-on-google

// // for a complete Dialogflow fulfillment library Actions on Google client library v2 integration sample

// Run the proper function handler based on the matched Dialogflow intent name

let intentMap = new Map();

intentMap.set('submit', submit);

// intentMap.set('your intent name here', yourFunctionHandler);

// intentMap.set('your intent name here', googleAssistantHandler);

agent.handleRequest(intentMap);

});

### GOOGLE EXAMPLE FULFILLMENT

// See https://github.com/dialogflow/dialogflow-fulfillment-nodejs

// for Dialogflow fulfillment library docs, samples, and to report issues

'use strict';

const functions = require('firebase-functions');

const {WebhookClient} = require('dialogflow-fulfillment');

const {Card, Suggestion} = require('dialogflow-fulfillment');

process.env.DEBUG = 'dialogflow:debug'; // enables lib debugging statements

exports.dialogflowFirebaseFulfillment = functions.https.onRequest((request, response) => {

const agent = new WebhookClient({ request, response });

console.log('Dialogflow Request headers: ' + JSON.stringify(request.headers));

console.log('Dialogflow Request body: ' + JSON.stringify(request.body));

function welcome(agent) {

agent.add(`Welcome to my agent!`);

}

function fallback(agent) {

agent.add(`I didn't understand`);

agent.add(`I'm sorry, can you try again?`);

}

// // Uncomment and edit to make your own intent handler

// // uncomment `intentMap.set('your intent name here', yourFunctionHandler);`

// // below to get this function to be run when a Dialogflow intent is matched

// function yourFunctionHandler(agent) {

// agent.add(`This message is from Dialogflow's Cloud Functions for Firebase editor!`);

// agent.add(new Card({

// title: `Title: this is a card title`,

// imageUrl: 'https://developers.google.com/actions/images/badges/XPM\_BADGING\_GoogleAssistant\_VER.png',

// text: `This is the body text of a card. You can even use line\n breaks and emoji! 💁`,

// buttonText: 'This is a button',

// buttonUrl: 'https://assistant.google.com/'

// })

// );

// agent.add(new Suggestion(`Quick Reply`));

// agent.add(new Suggestion(`Suggestion`));

// agent.setContext({ name: 'weather', lifespan: 2, parameters: { city: 'Rome' }});

// }

// // Uncomment and edit to make your own Google Assistant intent handler

// // uncomment `intentMap.set('your intent name here', googleAssistantHandler);`

// // below to get this function to be run when a Dialogflow intent is matched

// function googleAssistantHandler(agent) {

// let conv = agent.conv(); // Get Actions on Google library conv instance

// conv.ask('Hello from the Actions on Google client library!') // Use Actions on Google library

// agent.add(conv); // Add Actions on Google library responses to your agent's response

// }

// // See https://github.com/dialogflow/dialogflow-fulfillment-nodejs/tree/master/samples/actions-on-google

// // for a complete Dialogflow fulfillment library Actions on Google client library v2 integration sample

// Run the proper function handler based on the matched Dialogflow intent name

let intentMap = new Map();

intentMap.set('Default Welcome Intent', welcome);

intentMap.set('Default Fallback Intent', fallback);

// intentMap.set('your intent name here', yourFunctionHandler);

// intentMap.set('your intent name here', googleAssistantHandler);

agent.handleRequest(intentMap);

});

### NEWEST FULFILLMENT

'use strict';

const http = require('http');

const Datastore = require('@google-cloud/datastore');

const projectId = 'helpdesk-104b6';

const datastore = Datastore({projectId: projectId});

const kind = 'ticket';

const functions = require('firebase-functions');

const {WebhookClient} = require('dialogflow-fulfillment');

const {Card, Suggestion} = require('dialogflow-fulfillment');

var query, tasks, results, task, taskKey,found;

process.env.DEBUG = 'dialogflow:debug';

exports.dialogflowFirebaseFulfillment = functions.https.onRequest((request, response) => {

const agent = new WebhookClient({ request, response });

console.log('Dialogflow Request headers: ' + JSON.stringify(request.headers));

console.log('Dialogflow Request body: ' + JSON.stringify(request.body));

function welcome(agent)

{

agent.add(`Welcome to my agent!`);

}

function fallback(agent)

{

var fff = returnName();

agent.add(`I didn't understand: `+JSON.stringify(fff));

}

function getName(agent)

{

query = datastore.createQuery('ticket').filter('ticketNumber', 26703);

datastore.runQuery(query).then(results => {

console.log("FOUND: ",results);

});

agent.add(`DONE NAME: `+results);

}

function returnName()

{

query = datastore.createQuery('ticket').filter('ticketNumber', 26703);

datastore.runQuery(query).then(results => {

console.log("SENT: ",results);

return results;

});

}

function search(agent)

{

query = datastore.createQuery('ticket').order('date');

datastore.runQuery(query).then(results => {

tasks = results[0];

console.log('Tasks:');

tasks.forEach(task => {

taskKey = task[datastore.KEY];

console.log(taskKey.id, task);

});

}).catch(err => {

console.error('ERROR:', err);

});

agent.add(`DONE LOGGING`);

}

function printCurrentKey(key)

{

agent.add("PRINT: "+ key);

}

function randomIntInc (low, high)

{

return Math.floor(Math.random() \* (high - low + 1) + low);

}

function getDate()

{

let currentdate = new Date();

return ((currentdate.getMonth()+1) + "/"

+ currentdate.getDate() + "/"

+ currentdate.getFullYear() + " @ "

+ currentdate.getHours() + ":"

+ currentdate.getMinutes() + ":"

+ currentdate.getSeconds());

}

function submit(agent)

{

let ticketDescription = request.body.queryResult['queryText'];

let username = request.body.queryResult.outputContexts[1].parameters['given-name.original'];

let phone\_number = request.body.queryResult.outputContexts[1].parameters['phone-number.original'];

const taskKey = datastore.key(kind);

let datetime = getDate();

let ticketnum = randomIntInc(11111,99999);

const task = {

key: taskKey,

data: {

date: datetime,

description: ticketDescription,

username: username,

phoneNumber: phone\_number,

ticketNumber: ticketnum

}

};

datastore.save(task)

.then(() => {

console.log(`Saved ${task.key.name}: ${task.data.description}`);

})

.catch(err => {

console.error('ERROR:', err);

});

agent.add( username + ", I have successfully logged your ticket, the ticket number is " + ticketnum + ". The helpdesk team will reach out to you with in 24 hours. Thank you!");

}

let intentMap = new Map();

intentMap.set('Default Welcome Intent', welcome);

intentMap.set('Default Fallback Intent', fallback);

intentMap.set('Submit Ticket - collect description', submit);

intentMap.set('search', search);

intentMap.set('getName', getName);

agent.handleRequest(intentMap);

});

### GOOGLE SHEETS FULFILLMENT

'use strict';

const functions = require('firebase-functions');

const {WebhookClient} = require('dialogflow-fulfillment');

var GoogleSpreadsheet = require('google-spreadsheet');

var spreadsheetID = '1sd-2dTP16PVSQnmB3fPiDEubuPiUKRwb3AjKDs-WjoA';

var doc = new GoogleSpreadsheet(spreadsheetID);

var async = require('async');

var sheet, filtered, toFind,vals;

process.env.DEBUG = 'dialogflow:debug';

exports.dialogflowFirebaseFulfillment = functions.https.onRequest((request, response) => {

const agent = new WebhookClient({ request, response });

console.log('Dialogflow Request headers: ' + JSON.stringify(request.headers));

console.log('Dialogflow Request body: ' + JSON.stringify(request.body));

function welcome(agent) {

agent.add(`Welcome to my agent!`);

}

function fallback(agent) {

agent.add(`I didn't understand`);

agent.add(`I'm sorry, can you try again?`);

}

function numberReq(agent){

async.series([setAuth, getInfoAndWorksheets,searchfor], function (err){

if( err ) {

console.log('Error: '+err);

}

});

}

function myFunction(agent) {

agent.add(`Done: `+JSON.stringify(filtered));

}

function setAuth(step){

var creds\_json = {

client\_email: 'dialogflow-ucgmtn@sheets-7e77e.iam.gserviceaccount.com',

private\_key: '-----BEGIN PRIVATE KEY-----\nMIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQClwpiUzOvnJmWD\nrRSH+v7iuMOjNWGbkhgdwAmFanXUh6tAFeBGtfhxBxuB9uWnEOIC4jWM4StdMIjA\nogtwq8rQoPRryt9fJ3viEH0ERh+O3AifxXntAnL02ORAdQtkHjdNjYpSNcGgyEMb\n0FTNbZ88L1touBDMOSCrGzkFWHEbc7TVH/hvVSScG4pZR9L5+hlEVd+0/7hAGOVz\n9w7R2o1Jdlg7BmADDjEqD7Ip5gTVdXQ2D9qN2MlUiV8NxxdXG/ulDEcqTbClmbkr\nXsArGrv4rciRBXAmCN5WmBajv7V5W9AHpLVH+ivP1PLI9VSie3vf7jQTdzvffyIk\nT7OFCi+zAgMBAAECggEAORGn3ac12vXlwnF/HZgFVEKfnJve2KRyaB+7R3NRURRK\nmfxOJJo3CJcVWegKxjuyTKLcI/5MIz/3/TZtMOP8plqb/FazE/wwkzLxD90itVk9\nIODo6vhKYtEfuK9zdgtPKATAqDA402XPcQ6He9ivl5WIZ6PjqKRMgoyTU6tQ+c2M\n7E0b+cDlJYwonamMhR+RfvdE4m/PN2Ci0E8SDpZ/YkUg2IrU88dD/B9izAevqRLT\nehQ3d+ObrLknVH+vX3KhvYi1sRynKKYCzbDm+MsUAmob77YTdRS9KKYGmOAQ+g92\nGcMzhXE67KhERGwQsPLc1B3BeYycx2Jkrp6icSoYIQKBgQDXKvV9GfuajkpceA4S\nBu2lXXFYuxDvxA3De4cqSi4NsKKdu9B/qzNLLJ35IYXfSSo+htZ8tvuwhMfcUQGX\npTdnmY0xv5/90E6mtL35GBkBfBZ5P7BPqoV9jfQ1a7czcJHC3E5c3kfobdeIb/TQ\nfkUIJe9tkgUJA4wsRwtEcOo34QKBgQDFN16EFSDp6jjVC2bOPqOFKI8VhDIUKg+L\nRWQYOaGkDyhSN1PP60BCJyK8ClDeERoJX+u0Tdtqwhj5evi58rvcpnOgp2Q0DtTS\nTnenMME3/lhwYw9k2ah28XuYB00lUi+IIFtZ7nv2xCJbqgM7cX/YOMntA+7CkIgo\nB4m7q6FKEwKBgFC2hLAQ9xjbF3nobx7ECpFlpAcs5QWL/b2+sM1kVjI8eEPDGEtL\n5UnlyryRnq5Lu9UIr5zI51En/fDfUkGYcFgkeZqXz2Rep6JRWpH+QCMWcTe6dGDu\nDnUystD3PxnEuGjoS1z65O6Dhb5ggO1U8IBF+rKXe9wNwJ87ZzadRL8BAoGAGqio\naxxcDaKuugXHiNuoyQg5h0QObPFelonzxN6NfhrabXqzJqj7T7XwmRCf69TN+zRv\nLTlVDrJcI+U+mAw63tHVUYMvJWZX2HHo9L2/cK0+ETsiypQtlJ6CAYQDJqdca39M\nOfTl5u6OYmux8igeiwDqMtLURJK7BQY31RmXPOcCgYEAyAGEsam9D1Tc+w0n7fUB\niVnoLcdjU6u5JE/kHjXB/Xhuqt0nYIQUzroIf/cg0aNQObPdyBEQt6KTbGqHHNiL\nBRI9BnMXtq5QC6628ispTGBP+dONa0vbuuWjKp3oKomhVWW6Bnk9pQN9iqSViCXU\n+EEYXbRyBviqlVcH4RwAQp4=\n-----END PRIVATE KEY-----\n'

};

doc.useServiceAccountAuth(creds\_json,step);

}

function getInfoAndWorksheets(step) {

doc.getInfo(function(err, info) {

console.log('Loaded doc: '+info.title+' by '+info.author.email);

sheet = info.worksheets[0];

console.log('sheet 1: '+sheet.title+' '+sheet.rowCount+'x'+sheet.colCount);

step();

});

}

function readRows(step){

sheet.getRows({

offset: 1,

limit: 20,

orderby: 'col2'

}, function( err, rows ){

console.log('Read '+rows.length+' rows');

step();

});

}

function searchfor(step){

sheet.getCells({

'min-row': 1,

'max-row': 10,

'min-col': 1,

'max-col': 2,

'return-empty': false

}, function(err, cells) {

var toFind = "sean";

filtered = cells.filter(function(cell) {

return cell.value === toFind;

});

console.log(filtered);

step();

});

agent.tell(filtered);

}

let intentMap = new Map();

intentMap.set('Default Welcome Intent', welcome);

intentMap.set('Default Fallback Intent', fallback);

intentMap.set('readSheet', numberReq);

intentMap.set('readSheetY', myFunction);

agent.handleRequest(intentMap);

});

## 

### FAQ FINAL FULFILLMENT

'use strict';

const http = require('http');

const Datastore = require('@google-cloud/datastore');

const projectId = 'faq-pfqfnb';

const datastore = Datastore({projectId: projectId});

const kind = 'ticket';

const functions = require('firebase-functions');

const {WebhookClient} = require('dialogflow-fulfillment');

const {Card, Suggestion} = require('dialogflow-fulfillment');

var query, tasks, result, task, taskKey,found,output;

process.env.DEBUG = 'dialogflow:debug';

const host = 'api.worldweatheronline.com';

const wwoApiKey = '85a6283a74d149cf84d173031180507';

exports.dialogflowFirebaseFulfillment = functions.https.onRequest((request, response) => {

const agent = new WebhookClient({ request, response });

console.log('Dialogflow Request headers: ' + JSON.stringify(request.headers));

console.log('Dialogflow Request body: ' + JSON.stringify(request.body));

function randomIntInc (low, high)

{

return Math.floor(Math.random() \* (high - low + 1) + low);

}

function getDate()

{

let currentdate = new Date();

return ((currentdate.getMonth()+1) + "/"

+ currentdate.getDate() + "/"

+ currentdate.getFullYear() + " @ "

+ currentdate.getHours() + ":"

+ currentdate.getMinutes() + ":"

+ currentdate.getSeconds());

}

function submit(agent)

{

let ticketDescription = request.body.queryResult['queryText'];

let username = request.body.queryResult.outputContexts[1].parameters['given-name.original'];

let phone\_number = request.body.queryResult.outputContexts[1].parameters['phone-number.original'];

const taskKey = datastore.key(kind);

let datetime = getDate();

let ticketnum = randomIntInc(11111,99999);

const task = {

key: taskKey,

data: {

date: datetime,

description: ticketDescription,

username: username,

phoneNumber: phone\_number,

ticketNumber: ticketnum

}

};

datastore.save(task)

.then(() => {

console.log(`Saved ${task.key.name}: ${task.data.description}`);

})

.catch(err => {

console.error('ERROR:', err);

});

agent.add( username + ", I have successfully logged your ticket, the ticket number is " + ticketnum + ". The helpdesk team will reach out to you with in 24 hours. Thank you!");

}

function questions(agent)

{

agent.add(`Examples of frequently asked questions:`);

agent.add(`Where do I leave my keys when checking out?`);

agent.add(`How do I log in to the Wi-Fi in the apartment?`);

agent.add(`Is there parking?`);

agent.add(`Are pets allowed?`);

}

function weather(agent)

{

let city = request.body.queryResult.parameters['geo-city']; // city is a required param

// Get the date for the weather forecast (if present)

let date = '';

if (request.body.queryResult.parameters['date']) {

date = request.body.queryResult.parameters['date'];

console.log('Date: ' + date);

}

callWeatherApi(city, date);

agent.add("Type 'determine' to see the weather results.")

}

function weatheryes(agent)

{

//response.json({ 'fulfillmentText': output });

agent.add(output);

}

function callWeatherApi (city, date)

{

return new Promise((resolve, reject) => {

// Create the path for the HTTP request to get the weather

let path = '/premium/v1/weather.ashx?format=json&num\_of\_days=1' +

'&q=' + encodeURIComponent(city) + '&key=' + wwoApiKey + '&date=' + date;

console.log('API Request: ' + host + path);

// Make the HTTP request to get the weather

http.get({host: host, path: path}, (response) => {

let body = ''; // var to store the response chunks

response.on('data', (d) => { body += d; }); // store each response chunk

response.on('end', () => {

// After all the data has been received parse the JSON for desired data

let response = JSON.parse(body);

let forecast = response['data']['weather'][0];

let location = response['data']['request'][0];

let conditions = response['data']['current\_condition'][0];

let currentConditions = conditions['weatherDesc'][0]['value'];

// Create response

output = `Current conditions in the ${location['type']}

${location['query']} are ${currentConditions} with a projected high of

${forecast['maxtempC']}°C or ${forecast['maxtempF']}°F and a low of

${forecast['mintempC']}°C or ${forecast['mintempF']}°F on

${forecast['date']}.`;

// Resolve the promise with the output text

console.log(output);

resolve(output);

});

response.on('error', (error) => {

console.log(`Error calling the weather API: ${error}`);

reject();

});

});

});

}

let intentMap = new Map();

intentMap.set('Submit Ticket - collect description', submit);

intentMap.set('weather',weather);

intentMap.set('questions', questions);

intentMap.set('weather - yes', weatheryes);

agent.handleRequest(intentMap);

});

### COMBINED FULFILLMENT

'use strict';

const http = require('http');

const Datastore = require('@google-cloud/datastore');

const projectId = 'faq-pfqfnb';

const datastore = Datastore({projectId: projectId});

const kind = 'ticket';

const functions = require('firebase-functions');

const {WebhookClient} = require('dialogflow-fulfillment');

const {Card, Suggestion} = require('dialogflow-fulfillment');

var query, tasks, result, task, taskKey,found,output;

process.env.DEBUG = 'dialogflow:debug';

const host = 'api.worldweatheronline.com';

const wwoApiKey = '85a6283a74d149cf84d173031180507';

exports.dialogflowFirebaseFulfillment = functions.https.onRequest((request, response) => {

const agent = new WebhookClient({ request, response });

console.log('Dialogflow Request headers: ' + JSON.stringify(request.headers));

console.log('Dialogflow Request body: ' + JSON.stringify(request.body));

function welcome(agent)

{

agent.add(`Welcome to my agent!`);

}

function fallback(agent)

{

var fff = returnName();

agent.add(`I didn't understand: `+JSON.stringify(fff));

}

function getName(agent)

{

query = datastore.createQuery('ticket').filter('ticketNumber', 26703);

datastore.runQuery(query).then(results => {

result=results;

console.log("FOUND: ",results);

});

}

function printgetName(agent)

{

agent.add(`DONE NAME: `+JSON.stringify(result));

}

function returnName()

{

query = datastore.createQuery('ticket').filter('ticketNumber', 26703);

datastore.runQuery(query).then(results => {

console.log("SENT: ",results);

return results;

});

}

function search(agent)

{

query = datastore.createQuery('ticket').order('date');

datastore.runQuery(query).then(results => {

tasks = results[0];

console.log('Tasks:');

tasks.forEach(task => {

taskKey = task[datastore.KEY];

console.log(taskKey.id, task);

});

}).catch(err => {

console.error('ERROR:', err);

});

agent.add(`DONE LOGGING`);

}

function printCurrentKey(key)

{

agent.add("PRINT: "+ key);

}

function randomIntInc (low, high)

{

return Math.floor(Math.random() \* (high - low + 1) + low);

}

function getDate()

{

let currentdate = new Date();

return ((currentdate.getMonth()+1) + "/"

+ currentdate.getDate() + "/"

+ currentdate.getFullYear() + " @ "

+ currentdate.getHours() + ":"

+ currentdate.getMinutes() + ":"

+ currentdate.getSeconds());

}

function submit(agent)

{

let ticketDescription = request.body.queryResult['queryText'];

let username = request.body.queryResult.outputContexts[1].parameters['given-name.original'];

let phone\_number = request.body.queryResult.outputContexts[1].parameters['phone-number.original'];

const taskKey = datastore.key(kind);

let datetime = getDate();

let ticketnum = randomIntInc(11111,99999);

const task = {

key: taskKey,

data: {

date: datetime,

description: ticketDescription,

username: username,

phoneNumber: phone\_number,

ticketNumber: ticketnum

}

};

datastore.save(task)

.then(() => {

console.log(`Saved ${task.key.name}: ${task.data.description}`);

})

.catch(err => {

console.error('ERROR:', err);

});

agent.add( username + ", I have successfully logged your ticket, the ticket number is " + ticketnum + ". The helpdesk team will reach out to you with in 24 hours. Thank you!");

}

function questions(agent)

{

agent.add(`Examples of frequently asked questions:`);

agent.add(`Where do I leave my keys when checking out?`);

agent.add(`How do I log in to the Wi-Fi in the apartment?`);

agent.add(`Is there parking?`);

agent.add(`Are pets allowed?`);

}

function weather(agent)

{

let city = request.body.queryResult.parameters['geo-city']; // city is a required param

// Get the date for the weather forecast (if present)

let date = '';

if (request.body.queryResult.parameters['date']) {

date = request.body.queryResult.parameters['date'];

console.log('Date: ' + date);

}

callWeatherApi(city, date);

agent.add("Type 'determine' to see the weather results.")

}

function weatheryes(agent)

{

//response.json({ 'fulfillmentText': output });

agent.add(output);

}

function callWeatherApi (city, date)

{

return new Promise((resolve, reject) => {

// Create the path for the HTTP request to get the weather

let path = '/premium/v1/weather.ashx?format=json&num\_of\_days=1' +

'&q=' + encodeURIComponent(city) + '&key=' + wwoApiKey + '&date=' + date;

console.log('API Request: ' + host + path);

// Make the HTTP request to get the weather

http.get({host: host, path: path}, (response) => {

let body = ''; // var to store the response chunks

response.on('data', (d) => { body += d; }); // store each response chunk

response.on('end', () => {

// After all the data has been received parse the JSON for desired data

let response = JSON.parse(body);

let forecast = response['data']['weather'][0];

let location = response['data']['request'][0];

let conditions = response['data']['current\_condition'][0];

let currentConditions = conditions['weatherDesc'][0]['value'];

// Create response

output = `Current conditions in the ${location['type']}

${location['query']} are ${currentConditions} with a projected high of

${forecast['maxtempC']}°C or ${forecast['maxtempF']}°F and a low of

${forecast['mintempC']}°C or ${forecast['mintempF']}°F on

${forecast['date']}.`;

// Resolve the promise with the output text

console.log(output);

resolve(output);

});

response.on('error', (error) => {

console.log(`Error calling the weather API: ${error}`);

reject();

});

});

});

}

let intentMap = new Map();

intentMap.set('Default Welcome Intent', welcome);

intentMap.set('Default Fallback Intent', fallback);

intentMap.set('Submit Ticket - collect description', submit);

intentMap.set('search', search);

intentMap.set('getName', getName);

intentMap.set('weather',weather);

intentMap.set('printgetName', printgetName);

intentMap.set('questions', questions);

intentMap.set('weather - yes', weatheryes);

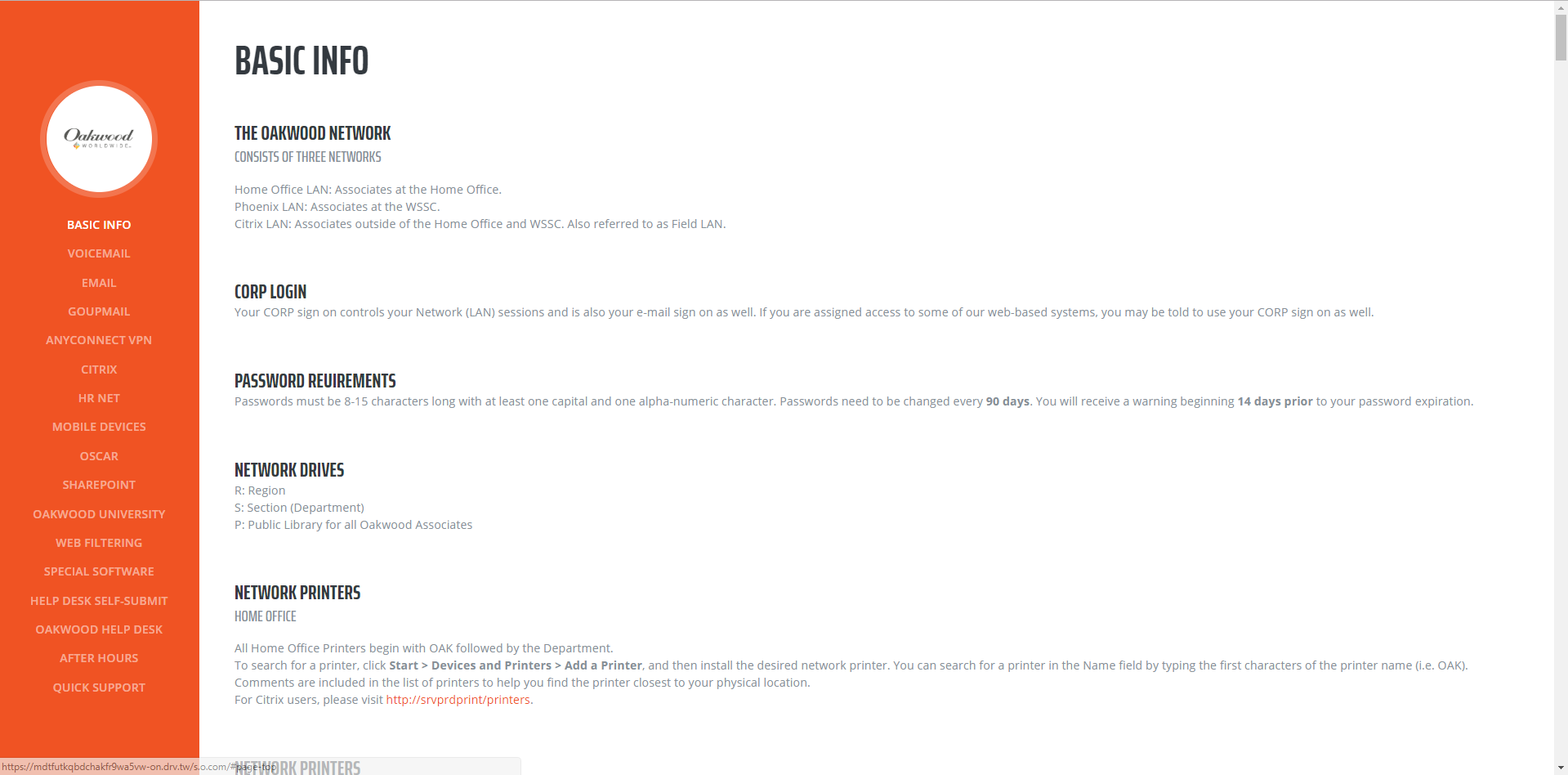
agent.handleRequest(intentMap);

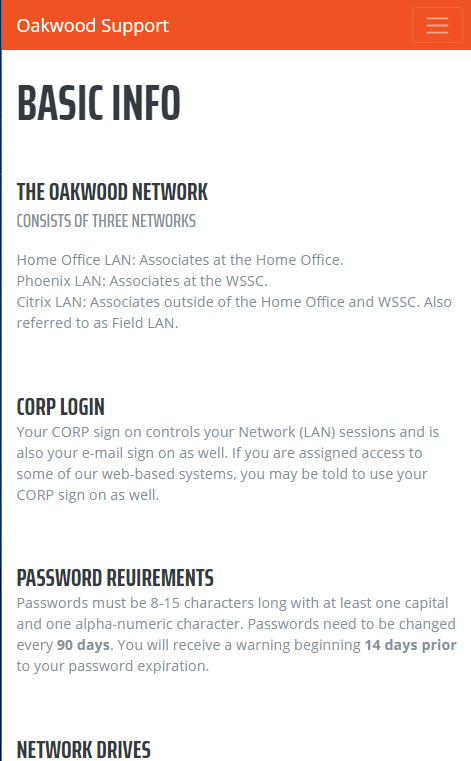
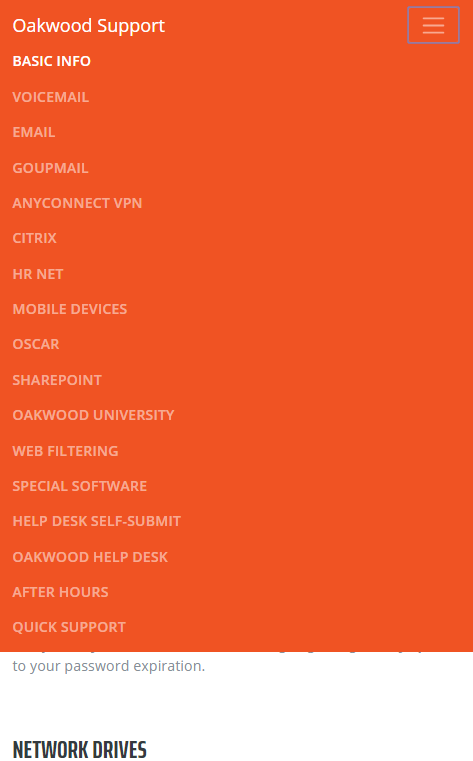
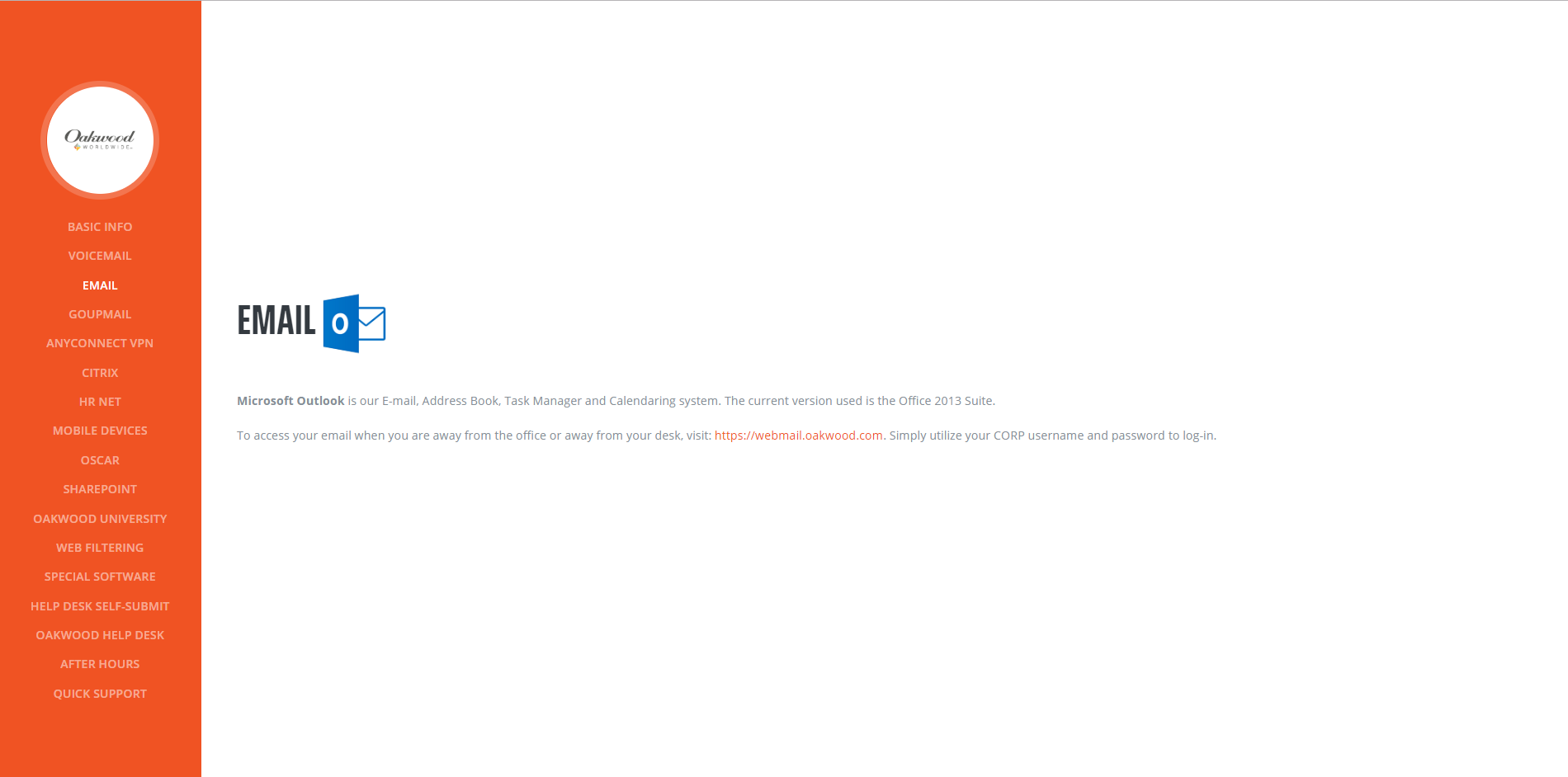
});

## RE-IMAGINE SUPPORT.OAKWOOD

## 

[Fake.support.oakwood.com](https://mdtfutkqbdchakfr9wa5vw-on.drv.tw/s.o.com/)





## PRICING SQL

Please, use User: [Access4Intern] with Password [7$v%sQuF]

For both server: DWPRDDB & DWQADB

SELECT

MARKET

, APT\_TYPE, COUNT(\*) REQUEST\_COUNT, SUM(QUOTED\_RATE\_USD) TOTAL\_QUOTED\_RATE

, MIN(QUOTED\_RATE\_USD) MIN\_RATE

, SUM(QUOTED\_RATE\_USD)/COUNT(\*) AVG\_RATE

, MAX(QUOTED\_RATE\_USD) MAX\_RATE

FROM

( SELECT

( CASE WHEN MAP.MAPPED = '' THEN D2.MARKET ELSE MAP.MAPPED END ) MARKET,

FR.QUOTED\_RATE\_USD,

FR.PROVIDER,

FR.APT\_TYPE APT\_TYPE\_SOURCE

, ( CASE WHEN APT\_TYPE LIKE '1%' THEN '1F'

WHEN APT\_TYPE LIKE '2%' THEN '2F'

WHEN APT\_TYPE LIKE '3%' THEN '3F'

WHEN APT\_TYPE LIKE '4%' THEN '4F'

WHEN APT\_TYPE LIKE 'S%' THEN 'SF'

ELSE 'UNK' END ) APT\_TYPE

, CREATED\_DATE

FROM OAKWOODBI.dbo.FACT\_RFH FR (NOLOCK)

INNER JOIN OAKWOODBI.dbo.DIM2\_RFH D2 (NOLOCK)

ON ( D2.EFFECT\_TO\_DT IS NULL AND D2.CUR\_RFH\_ID=FR.CUR\_RFH\_ID )

LEFT OUTER JOIN META\_EPIC\_CITY\_MAPPING MAP (NOLOCK)

ON D2.MARKET = MAP.MARKET

WHERE D2.MARKET Is Not Null

AND FR.QUOTED\_RATE\_USD Is Not Null

AND FR.QUOTED\_RATE\_USD <> 0

AND FR.APT\_TYPE Is Not Null

AND ULTIMATE\_STATUS = 'Booked'

AND CREATED\_DATE BETWEEN (DATEADD(YEAR, -1, GETDATE())) AND GETDATE() ) R

GROUP BY MARKET, APT\_TYPE

## YELP API

**Client ID**

LZYli2tEovy68EU\_zdwG5w

**API Key**

maeuatlGP-K3uflo7ryXhC9QdM2QxJT7gcKo\_mOEeHjobtFFIdLZfcM-iQoSyglGsU0Z6c1kMDwr2FyVX3kxulA\_8vJ2cyPBDT-N5\_2FMgNPWqtczdKxVXClcAtJW3Yx

// See https://github.com/dialogflow/dialogflow-fulfillment-nodejs

// for Dialogflow fulfillment library docs, samples, and to report issues

'use strict';

const yelp = require('yelp-fusion');

const functions = require('firebase-functions');

const {WebhookClient} = require('dialogflow-fulfillment');

const {Card, Suggestion} = require('dialogflow-fulfillment');

const apiKey = 'maeuatlGP-K3uflo7ryXhC9QdM2QxJT7gcKo\_mOEeHjobtFFIdLZfcM-iQoSyglGsU0Z6c1kMDwr2FyVX3kxulA\_8vJ2cyPBDT-N5\_2FMgNPWqtczdKxVXClcAtJW3Yx';

const client = yelp.client(apiKey);

var prettyJson = {};

var results = [];

var res = [];

process.env.DEBUG = 'dialogflow:debug'; // enables lib debugging statements

exports.dialogflowFirebaseFulfillment = functions.https.onRequest((request, response) => {

const agent = new WebhookClient({ request, response });

console.log('Dialogflow Request headers: ' + JSON.stringify(request.headers));

console.log('Dialogflow Request body: ' + JSON.stringify(request.body));

//get search keyword

let search = '';

if (request.body.queryResult.parameters['attraction']){

search = request.body.queryResult.parameters['attraction'];

} else {fallback(agent);}

//get search location: either zip code or city

//could use sys.location however API takes in a string

let city = '';

if (request.body.queryResult.parameters['geo-city']) {

city = request.body.queryResult.parameters['geo-city'];

console.log('City: ' + city);

} else if (request.body.queryResult.parameters['zip-code']){

city = request.body.queryResult.parameters['zip-code'];

console.log('Zip code ' + city);

} else {fallback(agent);}

//create search request

const searchRequest = {

term: search,

location: city

};

function welcome(agent) {

agent.add(`Welcome to my agent! Please search a keyword and a city or zipcode to see top Yelp results`);

}

function fallback(agent) {

agent.add(`Make sure your request has a keyword and a city`);

//agent.add(`I'm sorry, can you try again?`);

}

function myfunc(agent){

client.search(searchRequest).then(response => {

var i;

res = [];

results = [];

for (i = 0; i< 5; i++){

res.push(response.jsonBody.businesses[i]);

results.push(JSON.stringify(res[i]['name']));

}

// const firstResult = response.jsonBody.businesses[0];

// yname = JSON.stringify(firstResult['name']);

prettyJson = JSON.stringify(results, null, 4);

console.log(JSON.stringify(res, null, 4));

console.log(prettyJson);

//yname = JSON.stringify(prettyJson[0]['name']);

//console.log(yname);

}).catch(e => {console.log(e);})

}

function myfuncf(agent){

agent.add('Top 5 results are:');

for (var i=0; i<5; i++){

agent.add(results[i] + ', rating: ' + res[i]['rating']+ ' \/ 5' + ', '+ res[i]['location']['display\_address']);

//agent.add(new Card({

//title: 'Result'+ i.toString(),

//imageUrl: JSON.stringify(res[i]['image\\_url']),

//text: JSON.stringify(res[i]['name']),

// buttonText: 'Yelp link',

// buttonUrl: JSON.stringify(res[i]['url'])

//}))

}

}

// Run the proper function handler based on the matched Dialogflow intent name

let intentMap = new Map();

intentMap.set('Default Welcome Intent', welcome);

intentMap.set('Default Fallback Intent', fallback);

intentMap.set('Attractions', myfunc);

intentMap.set('AttractionsP', myfuncf);

intentMap.set('AttractionsF', myfunc);

intentMap.set('AttractionsFY', myfuncf);

agent.handleRequest(intentMap);

});

intentMap.set('Default Welcome Intent', welcome);

## Questions Cheat Sheet

Facebook (@OakwoodWorldwideFAQ): <https://www.facebook.com/OakwoodWorldwideFAQ/>

Web: <https://bot.dialogflow.com/OakwoodFAQ>

**Frequently Asked Questions:**

* How is an Oakwood Worldwide apartment different from a hotel?
* What is the difference between Oakwood and ExecuStay?
* What is required in order to make a reservation?
* Can I earn Marriott Rewards Points for my stay?
* Can I make a reservation and move in on that same day?
* Are pets allowed in your apartments?
* What is included in the daily rate?
* Is housekeeping included?
* What happens after I make a reservation?
* How do I book an apartment under my company’s corporate account?
* What are my payment options?
* How do I check in and access my apartment on arrival?
* Where do I leave my keys when checking out?
* How do I log in to the Wi-Fi in the apartment?
* Who do I contact for questions or service requests?
* Where are Oakwood Worldwide’s apartments located?
* What types of furnished and serviced apartments are available?
* What are your prices?
* Can I book a serviced apartment for a short term stay?
* If a colleague and I are traveling at the same time, can we share an apartment?
* What is check in and check out time?
* What kind of property amenities are available?
* What kind of property amenities are available?
* Do you offer insurance coverage while traveling?
* Is parking included?
* If my project timeline or assignment is extended, can I extend my stay?
* What if I need assistance operating the appliances and electronics in my apartment?

|  |  |
| --- | --- |
| **Helpdesk Ticket:**   * help * Ticket * I have a problem * I want to submit a ticket * Issue * Problem * Submit Ticket | **Reservation:**   * Can I rent a room in London for 2 weeks? * Rent * Can I rent a single from Miami * Please book me a 3 bedroom in SF * Reserve a studio |
| **Pricing:**   * How much does a studio in london cost? * What is the price in Miami for a Double? * 2 bedroom in London | **Weather:**   * What is the weather in Culver City CA?   **Yelp Search:**   * Things to do in Seattle * Restaurants 90024 (zip code) |

## FLASK SERVER

**App.py**

#!/usr/bin/env python

import urllib

import json

import os

from flask import Flask

from flask import request

from flask import make\_response

# Flask app should start in global layout

app = Flask(\_\_name\_\_)

@app.route('/webhook', methods=['POST'])

def webhook():

req = request.get\_json(silent=True, force=True)

print("Request:")

print(json.dumps(req, indent=4))

result = req.get("queryResult")

parameters = result.get("parameters")

zone = parameters.get("bank-name")

cost = {'Andhra Bank':'6.85%', 'Allahabad Bank':'6.75%', 'Axis Bank':'6.5%', 'Bandhan bank':'7.15%', 'Bank of Maharashtra':'6.50%', 'Bank of Baroda':'6.90%', 'Bank of India':'6.60%', 'Bharatiya Mahila Bank':'7.00%', 'Canara Bank':'6.50%', 'Central Bank of India':'6.60%', 'City Union Bank':'7.10%', 'Corporation Bank':'6.75%', 'Citi Bank':'5.25%', 'DBS Bank':'6.30%', 'Dena Bank':'6.80%', 'Deutsche Bank':'6.00%', 'Dhanalakshmi Bank':'6.60%', 'DHFL Bank':'7.75%', 'Federal Bank':'6.70%', 'HDFC Bank':'5.75% to 6.75%', 'Post Office':'7.10%', 'Indian Overseas Bank':'6.75%', 'ICICI Bank':'6.25% to 6.9%', 'IDBI Bank':'6.65%', 'Indian Bank':'4.75%', 'Indusind Bank':'6.85%', 'J&K Bank':'6.75%', 'Karnataka Bank':'6.50 to 6.90%', 'Karur Vysya Bank':'6.75%', 'Kotak Mahindra Bank':'6.6%', 'Lakshmi Vilas Bank':'7.00%', 'Nainital Bank':'7.90%', 'Oriental Bank of Commerce':'6.85%', 'Punjab National Bank':'6.75%', 'Punjab and Sind Bank':'6.4% to 6.80%', 'Saraswat bank':'6.8%', 'South Indian Bank':'6% to 6.75%', 'State Bank of India':'6.75%', 'Syndicate Bank':'6.50%', 'Tamilnad Mercantile Bank Ltd':'6.90%', 'UCO bank':'6.75%', 'United Bank Of India':'6%', 'Vijaya Bank':'6.50%', 'Yes Bank':'7.10%'}

speech = "The interest rate of " + zone + " is " + str(cost[zone])

print("Response:")

print(speech)

res = {

"fulfillmentText": speech,

"source": "BankRates"

}

res = json.dumps(res, indent=4)

print(res)

r = make\_response(res)

r.headers['Content-Type'] = 'application/json'

return r

if \_\_name\_\_ == '\_\_main\_\_':

port = int(os.getenv('PORT', 8080))

print ("Starting app on port %d" %(port))

app.run(debug=True, port=port, host='0.0.0.0')

**App.yaml**

runtime: python

env: flex

entrypoint: gunicorn -b :$PORT app:app

runtime\_config:

python\_version: 3

manual\_scaling:

instances: 1

resources:

cpu: 1

memory\_gb: 0.5

disk\_size\_gb: 10

**Requirements.txt**

Flask==0.10.1

google-api-python-client==1.6.6

gunicorn==19.7.1

if req.get("queryResult").get("action") == "api":

scope = ['https://spreadsheets.google.com/feeds']

creds = ServiceAccountCredentials.from\_json\_keyfile\_name('client\_secret.json', scope)

client = gspread.authorize(creds)

sheet = client.open("pyserver").sheet1

list\_of\_hashes = sheet.get\_all\_records()

print(list\_of\_hashes)

speech = "This is the api test."

else:

## DOCUMENT API CALL

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.IO;

while (true)

{

Console.WriteLine("Type res number: ");

int resident = Int32.Parse(Console.ReadLine());

Console.WriteLine("Type ten number: ");

int tenant = Int32.Parse(Console.ReadLine());

Console.WriteLine("");

string sURL;

sURL = "http://oscarqa/OscarAPI/api/echosign/getDocument/" + resident + "-" + tenant;

WebRequest wrGETURL;

wrGETURL = WebRequest.Create(sURL);

wrGETURL.Proxy = WebProxy.GetDefaultProxy();

Stream objStream;

objStream = wrGETURL.GetResponse().GetResponseStream();

StreamReader objReader = new StreamReader(objStream);

string sLine = "";

int i = 0;

while (sLine != null)

{

i++;

sLine = objReader.ReadLine();

if (sLine != null)

Console.WriteLine("{0}:{1}", i, sLine);

}

Console.WriteLine("");

Console.WriteLine("Try again? (y/n)");

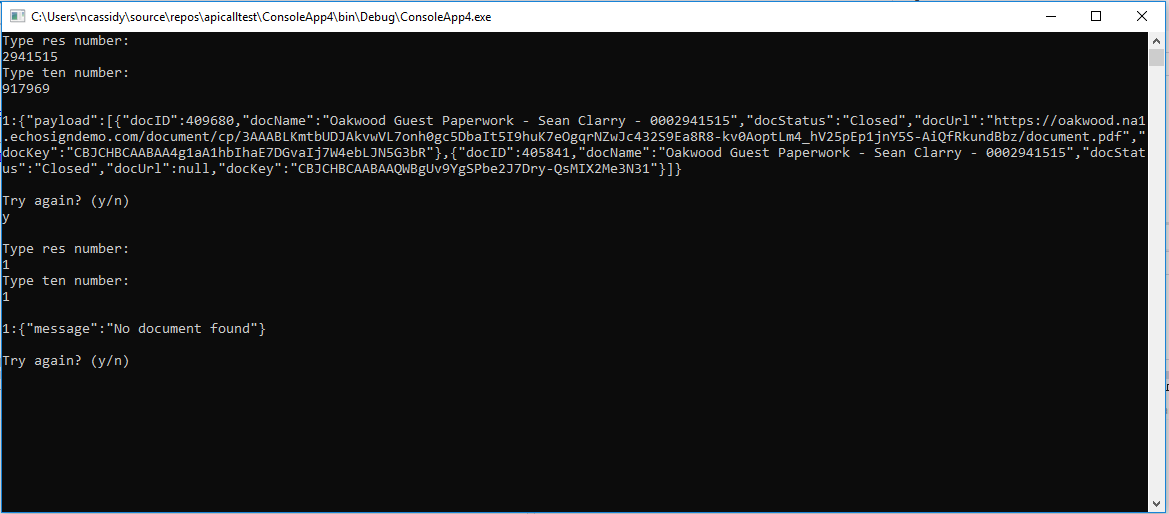
if (Console.ReadLine().ToLower() != "y")

break;

Console.WriteLine("");

}

}



## Python Search csv

import csv

datafile = open('Test.txt', 'r')

myreader = csv.reader(datafile)

#id 0,market 1,apt\_type 2,request\_count 3,total\_quoted\_rate 4,min\_rate 5,avg\_rate 6,max\_rate 7,last\_update 8

#row[0] = all ids

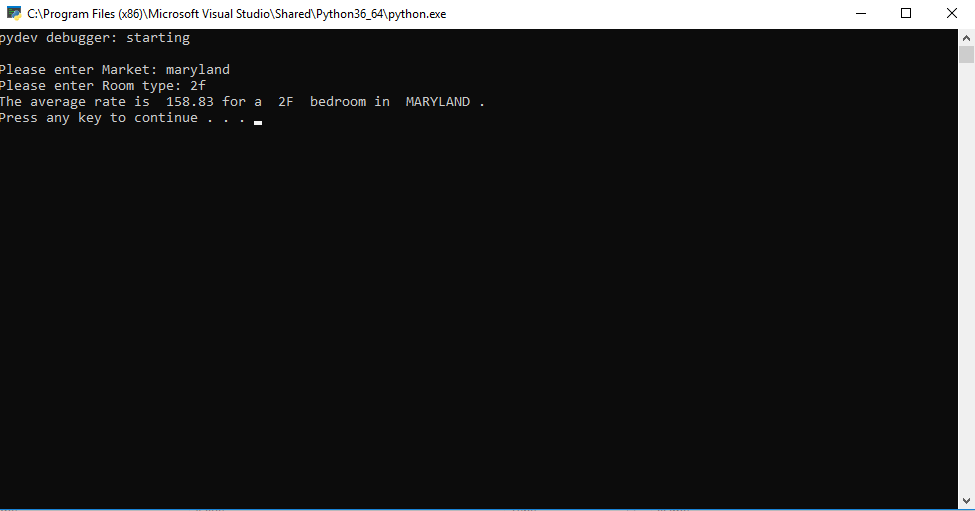
market = input("Please enter Market: ")

type = input("Please enter Room type: ")

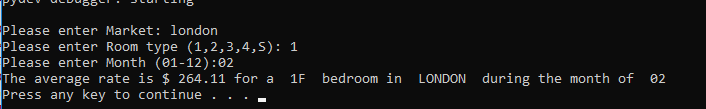
for row in myreader:

if(row[1].lower()==market.lower() and row[2].lower()==type.lower()):

print("The average rate is ",row[6],"for a ", row[2]," bedroom in ", row[1],".")



import csv  
import sqlite3  
datafile = open('TopCity.txt', 'r')  
myreader = csv.reader(datafile)  
date\_dict = {"01":"201801", "02":"201802", "03":"201803",  
 "04":"201804", "05":"201805", "06":"201806",   
 "07":"201707", "08":"201708", "09":"201709",   
 "10":"201710", "11":"201711", "12":"201712"}   
market = input("Please enter Market: ")  
type = input("Please enter Room type (1,2,3,4,S): ")  
type = type + "F"  
date = input("Please enter Month (01-12):")  
date = date\_dict[date]  
for row in myreader:  
 if(market.lower() in row[0].lower() and row[2].lower()==type.lower() and row[1].lower()==date.lower()):  
 print("The average rate is $", str(round(float(row[7]), 2)),"for a ", row[2]," bedroom in ", row[0]," during the month of ",date[-2:])



import csv

import calendar

datafile = open('TopCity.txt', 'r')

myreader = csv.reader(datafile)

date\_dict = {"01":"201801", "02":"201802", "03":"201803",

"04":"201804", "05":"201805", "06":"201806",

"07":"201707", "08":"201708", "09":"201709",

"10":"201710", "11":"201711", "12":"201712"}

market = input("Please enter Market: ")

type = input("Please enter Room type (1,2,3,4,S): ")

type = type + "F"

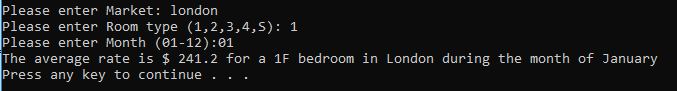
date = input("Please enter Month (01-12):")

date = date\_dict[date]

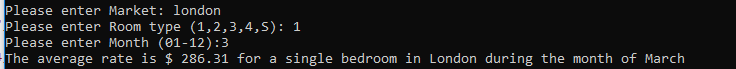
for row in myreader:

if(market.lower() in row[0].lower() and row[2].lower()==type.lower() and row[1].lower()==date.lower()):

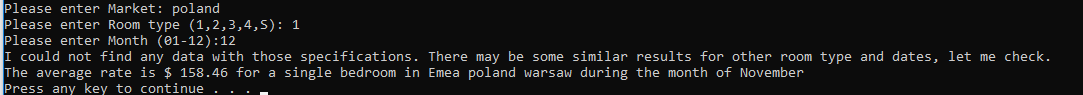
print("The average rate is $", str(round(float(row[7]), 2)),"for a", row[2],"bedroom in", row[0].capitalize(),"during the month of",calendar.month\_name[int(date[-2:])])



import csv  
import calendar  
  
datafile = open('TopCity.txt', 'r')  
myreader = csv.reader(datafile)  
types={"1":"single","2":"double","3":"triple", "4":"four", "S":"studio"}  
market = input("Please enter Market: ")  
type = input("Please enter Room type (1,2,3,4,S): ")  
date = input("Please enter Month (01-12):")  
for row in myreader:  
 if(market.lower() in row[0].lower() and row[2][0].lower()==type.lower() and int(row[1][-2:])==int(date)):  
 print("The average rate is $", str(round(float(row[7]), 2)),"for a", types[row[2][0]],"bedroom in", row[0].capitalize(),"during the month of",calendar.month\_name[int(date[-2:])])



import csv  
import calendar  
  
found = False  
datafile = open('TopCity.txt', 'r')  
myreader = list(csv.reader(datafile))  
types={"1":"single","2":"double","3":"triple", "4":"four", "S":"studio"}  
market = input("Please enter Market: ")  
type = input("Please enter Room type (1,2,3,4,S): ")  
date = input("Please enter Month (01-12):")  
for row in myreader:  
 if(market.lower() in row[0].lower() and row[2][0].lower()==type.lower() and int(row[1][-2:])==int(date)):  
 found = True  
 print("The average rate is $", str(round(float(row[7]), 2)),"for a", types[row[2][0]],"bedroom in", row[0].capitalize(),"during the month of",calendar.month\_name[int(date[-2:])])  
if(found == False):  
 print("I could not find any data with those specifications. There may be some similar results for other room type and dates, let me check.")  
 date = str(int(date)+1)  
 for row in myreader:  
 if(market.lower() in row[0].lower() and row[2][0].lower()==type.lower() and int(row[1][-2:])==int(date)):  
 found = True  
 print("The average rate is $", str(round(float(row[7]), 2)),"for a", types[row[2][0]],"bedroom in", row[0].capitalize(),"during the month of",calendar.month\_name[int(int(row[1][-2:]))])  
 date = str(int(date)-2)  
 for row in myreader:  
 if(market.lower() in row[0].lower() and row[2][0].lower()==type.lower() and int(row[1][-2:])==int(date)):  
 found = True  
 print("The average rate is $", str(round(float(row[7]), 2)),"for a", types[row[2][0]],"bedroom in", row[0].capitalize(),"during the month of",calendar.month\_name[int(int(row[1][-2:]))])



#check exact room type for yearly avg, print only if no relative monthly results

if (found==False):

for row in myreader:

if(market.lower() in row[0].lower() and row[2][0].lower()==type.lower()):

found = True

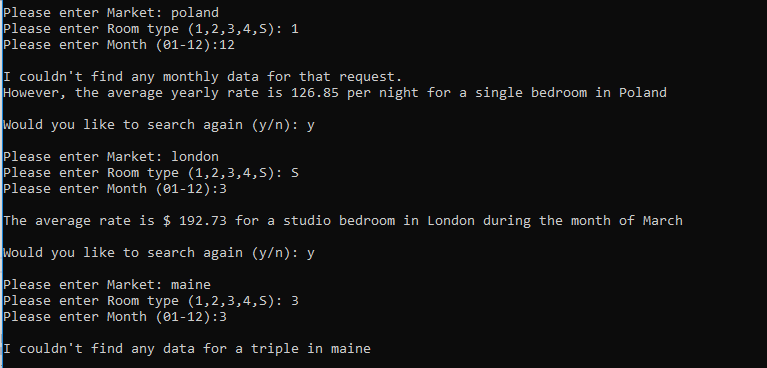
requestct += int(row[3])

totqtrate += float(row[4])

if (found == True):

print("I couldn't find any monthly data for that request.")

print("However, the average rate is", str(round(float(totqtrate/requestct),2)), "per night for a", types[type], "bedroom in", market.capitalize())



import csv

import calendar

datafile = open('TopCity.txt', 'r')

myreader = list(csv.reader(datafile))

types={"1":"single","2":"double","3":"triple", "4":"four", "S":"studio"}

while(True):

requestct = 0

totqtrate = 0

found = False

market = input("\nPlease enter Market: ")

type = input("Please enter Room type (1,2,3,4,S): ")

date = input("Please enter Month (01-12):")

#look for average monthly rate for given type

for row in myreader:

if(market.lower() in row[0].lower() and row[2][0].lower()==type.lower() and int(row[1][-2:])==int(date)):

found = True

print("\nThe average rate is $", str(round(float(row[7]), 2)),"for a", types[row[2][0]],"bedroom in", row[0].capitalize(),"during the month of",calendar.month\_name[int(date[-2:])])

#look for average yearly rate for given type

if (found==False):

for row in myreader:

if(market.lower() in row[0].lower() and row[2][0].lower()==type.lower()):

found = True

requestct += int(row[3])

totqtrate += float(row[4])

if (found == True):

print("\nI couldn't find any monthly data for that request.")

print("However, the average yearly rate is", str(round(float(totqtrate/requestct),2)), "per night for a", types[type], "bedroom in", market.capitalize())

if(found==False):

print("\nI couldn't find any data for a",types[type],"in",market,"\n")

cont = input("\nWould you like to search again (y/n): ")

if(cont.lower()=="n"):

break

## URL Python Search (MVS)

import csv

import calendar

from urllib.request import urlopen

with urlopen("https://storage.googleapis.com/topcitymonthlybucket1/TopCity.csv") as url:

cr = url.read()

splitcr = cr.splitlines()

found = False

i=0

while found==False and i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8')

if("LONDON" in check):

found=True

final = check

except:

print('weird character')

print(final)

**NEW**

with urlopen("https://storage.googleapis.com/topcitymonthlybucket1/TopCity.csv") as url:

cr = url.read()

splitcr = cr.splitlines()

types={"1":"single","2":"double","3":"triple", "4":"four", "S":"studio"}

found = False

i=0

while found==False and i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8')

if(zone.lower() in check.lower() and ("2018"+month in check or "2017"+month in check) and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

avgrate = checkarray[6]

res = {

"fulfillmentText": ("The average price is "+avgrate+" for a "+types[roomtype[0]]+" in "+market.capitalize() + " during " + calendar.month\_name[int(month)]),

"source": "Pricing"

}

res = json.dumps(res, indent=4)

print(res)

r = make\_response(res)

r.headers['Content-Type'] = 'application/json'

return r

found=True

except:

print('weird character')

## WORKING FLASK URL SEARCH

<https://bot.dialogflow.com/nolanpyserver>

#!/usr/bin/env python

import urllib

import json

import os

import csv

import calendar

from flask import Flask

from flask import request

from flask import make\_response

from urllib.request import urlopen

app = Flask(\_\_name\_\_)

@app.route('/webhook', methods=['POST'])

def webhook():

req = request.get\_json(silent=True, force=True)

print("Request:")

print(json.dumps(req, indent=4))

result = req.get("queryResult")

parameters = result.get("parameters")

zone = parameters.get("room-location")

month = parameters.get("room-month")

type = parameters.get("room-type")

print("Response:")

with urlopen("https://storage.googleapis.com/topcitymonthlybucket1/TopCity.csv") as url:

cr = url.read()

splitcr = cr.splitlines()

types={"1":"single","2":"double","3":"triple", "4":"four", "S":"studio"}

found = False

season = False

i=0

requestct =0

totqtrate =0

while found==False and i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8')

if(month.lower() == "summer" or month.lower() == "winter" or month.lower() == "fall" or month.lower() == "spring"):

season=True

if(month.lower()=="summer"):

if(zone.lower() in check.lower() and ("201806" in check or "201707" in check or "201708" in check) and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

if(month.lower()=="spring"):

if(zone.lower() in check.lower() and ("201803" in check or "201804" in check or "201805" in check) and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

if(month.lower()=="fall"):

if(zone.lower() in check.lower() and ("201709" in check or "201710" in check or "201711" in check) and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

if(month.lower()=="winter"):

if(zone.lower() in check.lower() and ("201801" in check or "201802" in check or "201712" in check) and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

else:

if(zone.lower() in check.lower() and ("2018"+month in check or "2017"+month in check) and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

avgrate = checkarray[6]

res = {

"fulfillmentText": ("The average price is $"+str(round(float(avgrate), 2))+" for a "+types[roomtype[0]]+" in "+market.capitalize() + " during " + calendar.month\_name[int(month)]),

"source": "Pricing"

}

res = json.dumps(res, indent=4)

print(res)

r = make\_response(res)

r.headers['Content-Type'] = 'application/json'

return r

found=True

if(zone.lower() in check.lower() and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

avgrate = checkarray[6]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

except:

print('weird character')

if(requestct!=0 and totqtrate!=0 and season==True):

returnstring="The average price is $"+ str(round(float(totqtrate/requestct),2))+ " for a "+ types[roomtype[0]]+" in "+market.capitalize() + " during " + month.capitalize()

if(requestct!=0 and totqtrate!=0 and season==False):

returnstring="I could not find monthly data for that request. However, the average yearly rate is $"+ str(round(float(totqtrate/requestct),2))+ " for a "+ types[roomtype[0]]+ " in "+ market.capitalize()

if(found==False and requestct==0 and totqtrate==0):

returnstring="Sorry, I could not find any pricing for that search request. Please try again with a different city. "

res = {

"fulfillmentText": returnstring,

"source": "no Pricing"

}

res = json.dumps(res, indent=4)

print(res)

r = make\_response(res)

r.headers['Content-Type'] = 'application/json'

return r

if \_\_name\_\_ == '\_\_main\_\_':

port = int(os.getenv('PORT', 8080))

print ("Starting app on port %d" %(port))

app.run(debug=True, port=port, host='0.0.0.0')

## NEWEST FLASK SERVER BY SEASON

<https://bot.dialogflow.com/OakwoodPricingProduction>

#!/usr/bin/env python

import urllib

import json

import os

import csv

import calendar

from flask import Flask

from flask import request

from flask import make\_response

from urllib.request import urlopen

app = Flask(\_\_name\_\_)

@app.route('/webhook', methods=['POST'])

def webhook():

print("Request:")

req = request.get\_json(silent=True, force=True)

print(json.dumps(req, indent=4))

result = req.get("queryResult")

parameters = result.get("parameters")

zone = parameters.get("room-location")

month = parameters.get("room-month")

type = parameters.get("room-type")

types={"1":"single","2":"double","3":"triple", "4":"four", "S":"studio"}

splitcr = getCSV()

found = False

if(month.lower()=="summer"):

returnstring,found = summer(splitcr,zone,month,type,types)

elif(month.lower()=="spring"):

returnstring,found = spring(splitcr,zone,month,type,types)

elif(month.lower()=="fall"):

returnstring,found = fall(splitcr,zone,month,type,types)

elif(month.lower()=="winter"):

returnstring,found = winter(splitcr,zone,month,type,types)

elif(month.lower()=="year"):

returnstring,found = year(splitcr,zone,month,type,types)

else:

returnstring,found = monthly(splitcr,zone,month,type,types)

if(found == False):

returnstring,found = yearly(splitcr,zone,month,type,types)

if(found == False):

returnstring="Sorry, I could not find any pricing for that search request. Please try again with a different city. "

print("Response:")

res = {

"fulfillmentText": returnstring,

"source": "Pricing"

}

res = json.dumps(res, indent=4)

print(res)

r = make\_response(res)

r.headers['Content-Type'] = 'application/json'

return r

def summer(splitcr,zone,month,type,types):

i = 0

found = False

requestct = 0

totqtrate = 0

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8')

if(zone.lower() in check.lower() and ("201806" in check or "201707" in check or "201708" in check) and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

found=True

except:

print('weird character')

if(found == True):

returnstring="The average price is $"+ str(round(float(totqtrate/requestct),2))+ " for a "+ types[roomtype[0]]+" in "+market.title() + " during the " + month.capitalize()

else:

returnstring=""

return returnstring,found

def spring(splitcr,zone,month,type,types):

i = 0

found = False

requestct = 0

totqtrate = 0

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8')

if(zone.lower() in check.lower() and ("201803" in check or "201804" in check or "201805" in check) and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

found=True

except:

print('weird character')

if(found == True):

returnstring="The average price is $"+ str(round(float(totqtrate/requestct),2))+ " for a "+ types[roomtype[0]]+" in "+market.title() + " during the " + month.capitalize()

else:

returnstring=""

return returnstring,found

def fall(splitcr,zone,month,type,types):

i = 0

found = False

requestct = 0

totqtrate = 0

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8')

if(zone.lower() in check.lower() and ("201709" in check or "201710" in check or "201711" in check) and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

found=True

except:

print('weird character')

if(found == True):

returnstring="The average price is $"+ str(round(float(totqtrate/requestct),2))+ " for a "+ types[roomtype[0]]+" in "+market.title() + " during the " + month.capitalize()

else:

returnstring=""

return returnstring,found

def winter(splitcr,zone,month,type,types):

i = 0

found = False

requestct = 0

totqtrate = 0

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8')

if(zone.lower() in check.lower() and ("201801" in check or "201802" in check or "201712" in check) and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

found=True

except:

print('weird character')

if(found == True):

returnstring="The average price is $"+ str(round(float(totqtrate/requestct),2))+ " for a "+ types[roomtype[0]]+" in "+market.title() + " during the " + month.capitalize()

else:

returnstring=""

return returnstring,found

def monthly(splitcr,zone,month,type,types):

i = 0

found = False

while found==False and i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8')

if(zone.lower() in check.lower() and ("2018"+month in check or "2017"+month in check) and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

avgrate = checkarray[6]

found=True

except:

print('weird character')

if(found == True):

returnstring="The average price is $"+ str(round(float(avgrate),2))+ " for a "+ types[roomtype[0]]+" in "+market.title() + " during " + calendar.month\_name[int(month)]

else:

returnstring=""

return returnstring,found

def yearly(splitcr,zone,month,type,types):

i = 0

found = False

requestct = 0

totqtrate = 0

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8')

if(zone.lower() in check.lower() and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

avgrate = checkarray[6]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

found=True

except:

print('weird character')

if(found == True):

returnstring="I could not find monthly data for that request. However, the average yearly rate is $"+ str(round(float(totqtrate/requestct),2))+ " for a "+ types[roomtype[0]]+ " in "+ market.title()

else:

returnstring=""

return returnstring,found

def year(splitcr,zone,month,type,types):

i = 0

found = False

requestct = 0

totqtrate = 0

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8')

if(zone.lower() in check.lower() and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

avgrate = checkarray[6]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

found=True

except:

print('weird character')

if(found == True):

returnstring="The average yearly rate is $"+ str(round(float(totqtrate/requestct),2))+ " for a "+ types[roomtype[0]]+ " in "+ market.title()

else:

returnstring=""

return returnstring,found

def getCSV():

with urlopen("https://storage.googleapis.com/topcitymonthlybucket1/TopCity.csv") as url:

cr = url.read()

splitcr = cr.splitlines()

return splitcr

if \_\_name\_\_ == '\_\_main\_\_':

port = int(os.getenv('PORT', 8080))

print ("Starting app on port %d" %(port))

app.run(debug=True, port=port, host='0.0.0.0')

## FAQ - OAKWOOD LOCATION LINK PYTHON

#!/usr/bin/env python

import urllib

import json

import os

from geopy.geocoders import Nominatim

from flask import Flask

from flask import request

from flask import make\_response

# Flask app should start in global layout

app = Flask(\_\_name\_\_)

@app.route('/webhook', methods=['POST'])

def webhook():

req = request.get\_json(silent=True, force=True)

print("Request:")

print(json.dumps(req, indent=4))

result = req.get("queryResult")

parameters = result.get("parameters")

indate = str(parameters.get("in-date"))[:10]

outdate = str(parameters.get("out-date"))[:10]

query = str(parameters.get("query"))

bedrooms = str(parameters.get("types"))

geolocator = Nominatim(user\_agent="ow\_faq\_bot")

location = geolocator.geocode(query)

print(location.address)

#print(location.raw)

lat, lng = location.latitude, location.longitude

print(lat, lng)

baseurl = 'https://www.oakwood.com/search?lat='

speech = 'Please follow the link below to view availability near '+ location.address + ': \n' + baseurl + str(lat) + '&lng=' + str(lng) + '&moveIn='+ indate + '&moveOut=' + outdate + '&rooms=' + bedrooms

speech += '\nThe fastest way to get pricing and booking information is by giving us a call at 1-866-400-1925.\nIs there anything else I can help you with?'

print("Response:")

print(speech)

res = {

"fulfillmentText": speech

}

res = json.dumps(res, indent=4)

print(res)

r = make\_response(res)

r.headers['Content-Type'] = 'application/json'

return r

if \_\_name\_\_ == '\_\_main\_\_':

port = int(os.getenv('PORT', 8080))

print ("Starting app on port %d" %(port))

app.run(debug=True, port=port, host='0.0.0.0')

WIP ADDING FB USER NAME (DOES NOT RUN)

'use strict';

const http = require('http');

const Datastore = require('datastore');

const projectId = 'faq-pfqfnb';

const datastore = Datastore({projectId: projectId});

const functions = require('firebase-functions');

const {WebhookClient} = require('dialogflow-fulfillment');

const {Card, Suggestion} = require('dialogflow-fulfillment');

var query, tasks, result, task, taskKey,found,output,prettyJson;

process.env.DEBUG = 'dialogflow:debug';

const host = 'api.worldweatheronline.com';

const wwoApiKey = '85a6283a74d149cf84d173031180507';

const yelp = require('yelp-fusion');

const apiKey = 'maeuatlGP-K3uflo7ryXhC9QdM2QxJT7gcKo\_mOEeHjobtFFIdLZfcM-iQoSyglGsU0Z6c1kMDwr2FyVX3kxulA\_8vJ2cyPBDT-N5\_2FMgNPWqtczdKxVXClcAtJW3Yx';

const client = yelp.client(apiKey);

var prettyJson = {};

var results = [];

var res = [];

exports.dialogflowFirebaseFulfillment = functions.https.onRequest((request, response) => {

const agent = new WebhookClient({ request, response });

console.log('Dialogflow Request headers: ' + JSON.stringify(request.headers));

console.log('Dialogflow Request body: ' + JSON.stringify(request.body));

let responseJson = {};

function randomIntInc (low, high)

{

return Math.floor(Math.random() \* (high - low + 1) + low);

}

function getDate()

{

let currentdate = new Date();

return ((currentdate.getMonth()+1) + "/"

+ currentdate.getDate() + "/"

+ currentdate.getFullYear() + " @ "

+ currentdate.getHours() + ":"

+ currentdate.getMinutes() + ":"

+ currentdate.getSeconds());

}

function fbWelcome(agent){

http.get({host: 'graph.facebook.com', path: '/v3.1/{personal-user-id}'}, (response) => {

let body = ''; // var to store the response chunks

response.on('data', (d) => { body += d; }); // store each response chunk

response.on('end', () => {

let response = JSON.parse(body);

// Create response

output = `Current conditions in the ${location['type']}

${location['query']} are ${currentConditions} with a projected high of

${forecast['maxtempC']}°C or ${forecast['maxtempF']}°F and a low of

${forecast['mintempC']}°C or ${forecast['mintempF']}°F on

${forecast['date']}.`;

// Resolve the promise with the output text

console.log(output);

resolve(output);

});

response.on('error', (error) => {

console.log(`Error calling the weather API: ${error}`);

reject();

});

});

});

}

'input.FACEBOOK\_WELCOME': () =>{

responseJson.speech = 'Hi,'

}

}

function questions(agent)

{

agent.add(`Examples of frequently asked questions:`);

agent.add(`Are pets allowed?`);

agent.add(`What is your customer service phone number?`);

//agent.add(`Ticket`);

//agent.add(`Reserve a room`);

agent.add(`What are your popular locations?`);

agent.add(`How do I log on to the Wi-Fi?`);

}

function weather(agent)

{

let city = request.body.queryResult.parameters['geo-city']; // city is a required param

// Get the date for the weather forecast (if present)

let date = '';

if (request.body.queryResult.parameters['date']) {

date = request.body.queryResult.parameters['date'];

console.log('Date: ' + date);

}

callWeatherApi(city, date);

agent.add("Type 'determine' to see the weather results.")

}

function weatheryes(agent)

{

//response.json({ 'fulfillmentText': output });

agent.add(output);

}

function callWeatherApi (city, date)

{

return new Promise((resolve, reject) => {

// Create the path for the HTTP request to get the weather

let path = '/premium/v1/weather.ashx?format=json&num\_of\_days=1' +

'&q=' + encodeURIComponent(city) + '&key=' + wwoApiKey + '&date=' + date;

console.log('API Request: ' + host + path);

// Make the HTTP request to get the weather

http.get({host: host, path: path}, (response) => {

let body = ''; // var to store the response chunks

response.on('data', (d) => { body += d; }); // store each response chunk

response.on('end', () => {

// After all the data has been received parse the JSON for desired data

let response = JSON.parse(body);

let forecast = response['data']['weather'][0];

let location = response['data']['request'][0];

let conditions = response['data']['current\_condition'][0];

let currentConditions = conditions['weatherDesc'][0]['value'];

// Create response

output = `Current conditions in the ${location['type']}

${location['query']} are ${currentConditions} with a projected high of

${forecast['maxtempC']}°C or ${forecast['maxtempF']}°F and a low of

${forecast['mintempC']}°C or ${forecast['mintempF']}°F on

${forecast['date']}.`;

// Resolve the promise with the output text

console.log(output);

resolve(output);

});

response.on('error', (error) => {

console.log(`Error calling the weather API: ${error}`);

reject();

});

});

});

}

function fallback(agent)

{

agent.add(`Make sure your request has a keyword and a city`);

}

function yelpsearch(agent)

{

let search = '';

if (request.body.queryResult.parameters['attraction']){

search = request.body.queryResult.parameters['attraction'];

} else {fallback(agent);}

//get search location: either zip code or city

//could use sys.location however API takes in a string

let city = '';

if (request.body.queryResult.parameters['geo-city']) {

city = request.body.queryResult.parameters['geo-city'];

console.log('City: ' + city);

} else if (request.body.queryResult.parameters['zip-code']){

city = request.body.queryResult.parameters['zip-code'];

console.log('Zip code ' + city);

} else {fallback(agent);}

//create search request

const searchRequest = {

term: search,

location: city

};

client.search(searchRequest).then(response => {

var i;

res = [];

results = [];

for (i = 0; i< 5; i++){

res.push(response.jsonBody.businesses[i]);

results.push(JSON.stringify(res[i]['name']));

}

prettyJson = JSON.stringify(results, null, 4);

console.log(JSON.stringify(res, null, 4));

console.log(prettyJson);

}).catch(e => {console.log(e);})

}

function yelpyes(agent)

{

agent.add('Top 5 Yelp results are:');

for (var i=0; i<5; i++){

agent.add(results[i] + ', rating: ' + res[i]['rating']+ ' \/ 5' + ', '+ res[i]['location']['display\_address']);

}

}

let intentMap = new Map();

intentMap.set('weather',weather);

intentMap.set('Menu', questions);

intentMap.set('weather - yes', weatheryes);

intentMap.set('Attractions', yelpsearch);

intentMap.set('AttractionsP', yelpyes);

intentMap.set('AttractionsF', yelpsearch);

intentMap.set('AttractionsFY', yelpyes);

## 

## Articles

10 Examples of companies using bots:

<https://www.inc.com/larry-kim/10-examples-of-how-brands-are-using-chatbots-to-de.html>

5 More:

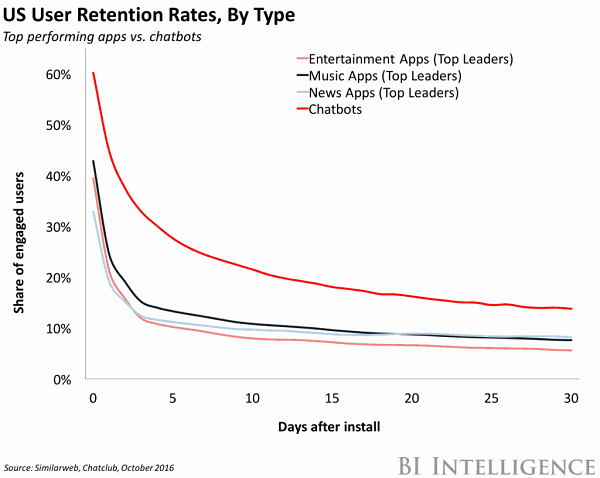
<https://squareup.com/townsquare/how-5-innovative-businesses-are-using-chatbots>

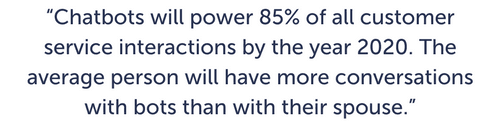
Great data:

<https://chatbotslife.com/how-businesses-are-winning-with-chatbots-ai-5df2f6304f81>

80% by 2020, Oracle study

<https://www.businessinsider.com/80-of-businesses-want-chatbots-by-2020-2016-12>

* Marriot already has 3 bots in production and working on many more
* 80% of companies want a bot by 2020 according to oracle
* Ecommerce/Marketting
  + 1–800-Flowers: reported that more than 70% of its Messenger orders derived from new customers!
  + Sephora: increased their makeover appointments by 11% via their Facebook Messenger Chatbot.
  + Nitro Café: increased sales by 20 percent with their Messenger chatbot which was designed for easy ordering, direct payments and instant two-way communication.
  + Sun’s Soccer: Chatbots drove nearly 50% of its users back to their site throughout specific soccer coverage; 43 percent of chatbot subscribers clicked through during their best period.
  + Asos: increased orders by 300% using Messenger Chatbots and got a 250% return on Spend while reaching 3.5x more people.
* Email
  + Messenger Chatbots have higher open rates and click through rates than email
* Sales funnel
  + If user says no you can redirect
* Relationship
  + develop a more personal relationship with users which increases customer loyalty
* Increased Retention Rates
  + bots are beating apps single handedly
  + transient, you don’t have to download them, sign up, etc…
  + this low barrier to entry and low friction makes it easier to use them when you want on your terms
  + it easy for the bot to contact you and start a conversation
* Customer support
  + Companies are saving as much as 29% on customer service by deploying bots, according to McKinsey
    - BI Intelligence estimates that chatbots will equate to a $23 billion in savings from annual salaries.
  + Additionally companies can automate 36% of sales representative positions
    - resulting in total annual estimated savings of at least $15 billion from salaries
  + IBM survey 65% of US millennials prefer going online to get support, rather than speak to staff in-store
  + Rodgers Wireless: had a 60% improvement in customer service
  + Globe Telecom: increased customer satisfaction by 22% while decreasing call volume by 50% using a Facebook Messenger Chatbot. Employees productivity increased by 350%.
* Hospitality
  + 1) They Increase Revenue 2) Increase customer satisfaction 3) Increase Engagement and Brand Loyalty and 4) Lower Costs via Automation
  + Give deals
  + Get pricing
* Finance
  + Chase: JPMorgan has numerous bots! COIN, has saved over 360,000 hours of manpower and its simple bots work by parsing emails for employees, grant access to software systems, and handle common IT requests like resetting passwords and more. Bots are expected to handle 1.7 million access requests this year, doing the work of 140 people.
  + HiCharlie: This bot keep track of your finances so you don’t have to. On average Charlie saves users $80 per week!
  + Sweedbank: Their chatbot, Nina, handles 40,000 conversations a month and resolves 81% of the issues.
  + Trim: Trim is Financial Assistant bot that can help you reduce expenses and even negotiate bills on your behalf. Trim has a 94% retention rate!
  + 

\

FINAL PRICING

**App.py**

#!/usr/bin/env python

import urllib

import json

import os

import csv

import calendar

from geopy.geocoders import Nominatim

from flask import Flask

from flask import request

from flask import make\_response

from urllib.request import urlopen

from fuzzywuzzy import fuzz

from forex\_python.converter import CurrencyRates

app = Flask(\_\_name\_\_)

@app.route('/webhook', methods=['POST'])

def webhook():

print("Request:")

req = request.get\_json(silent=True, force=True)

print(json.dumps(req, indent=4))

returnstring=""

result = req.get("queryResult")

action = result.get("action")

types={"S":"studio","1":"single","2":"double","3":"triple", "4":"four"}

if(action=="getprice"):

parameters = result.get("parameters")

zone = parameters.get("room-location")

month = parameters.get("room-month")

type = parameters.get("room-type")

currency = parameters.get("currency")

if(zone.lower() == "warszawa"):

zone = "warsaw"

splitcr = getCSV()

found = False

foundfuzzy=False

if(month.lower()=="summer"):

returnstring,found = summer(splitcr,zone,month,type,types,currency)

elif(month.lower()=="spring"):

returnstring,found = spring(splitcr,zone,month,type,types,currency)

elif(month.lower()=="fall"):

returnstring,found = fall(splitcr,zone,month,type,types,currency)

elif(month.lower()=="winter"):

returnstring,found = winter(splitcr,zone,month,type,types,currency)

elif(month.lower()=="year"):

returnstring,found = year(splitcr,zone,month,type,types,currency)

else:

returnstring,found = monthly(splitcr,zone,month,type,types,currency)

#checks for yearly with the same room type and location

if(found == False or returnstring==""):

returnstring,found = yearly(splitcr,zone,month,type,types,currency)

#checks for yearly with the same location

if(found == False or returnstring==""):

returnstring,found = yearlylast(splitcr,zone,month,type,types,currency)

if(found == False or returnstring==""):

foundzone,foundfuzzy = fuzzysearch(splitcr,zone,month,type,types)

if(foundfuzzy==True):

if(month.lower()=="summer"):

returnstring,found = summer(splitcr,foundzone,month,type,types,currency)

elif(month.lower()=="spring"):

returnstring,found = spring(splitcr,foundzone,month,type,types,currency)

elif(month.lower()=="fall"):

returnstring,found = fall(splitcr,foundzone,month,type,types,currency)

elif(month.lower()=="winter"):

returnstring,found = winter(splitcr,foundzone,month,type,types,currency)

elif(month.lower()=="year"):

returnstring,found = year(splitcr,foundzone,month,type,types,currency)

else:

returnstring,found = monthly(splitcr,foundzone,month,type,types,currency)

#checks for yearly with the same room type and location

if(found == False or returnstring==""):

returnstring,found = yearly(splitcr,foundzone,month,type,types,currency)

#checks for yearly with the same location

if(found == False or returnstring==""):

returnstring,found = yearlylast(splitcr,foundzone,month,type,types,currency)

returnstring="I think you misspelled the location \""+zone.title()+"\". Here is what I found for " + foundzone.title() + ". " + returnstring

if(found == False or returnstring==""):

returnstring="Sorry, I could not find any pricing for that search request. Please try again with a different city. "

elif(action == "getpricefollowup"):

outputcontext = result.get("outputContexts")[0]

parameters = outputcontext.get("parameters")

zone = parameters.get("room-location")

if(zone.lower() == "warszawa"):

zone = "warsaw"

month = parameters.get("room-month")

type = parameters.get("room-type")

extra = parameters.get("extras")

currency = parameters.get("currency")

splitcr = getCSV()

found = False

if(extra=="peak"):

svm = parameters.get("seasonvsmonth")

if(svm=="month"):

returnstring = peakmonthnew(splitcr,zone,month,type,types,currency)

elif(svm=="season" or svm==""):

returnstring = peakseason(splitcr,zone,month,type,types,currency)

elif(extra=="low"):

svm = parameters.get("seasonvsmonth")

if(svm=="month"):

returnstring = lowmonthnew(splitcr,zone,month,type,types,currency)

elif(svm=="season" or svm==""):

returnstring = lowseason(splitcr,zone,month,type,types,currency)

elif(action =="getavail"):

outputcontext = result.get("outputContexts")[0]

parameters = outputcontext.get("parameters")

zone = parameters.get("room-location")

if(zone.lower() == "warszawa"):

zone = "warsaw"

month = parameters.get("room-month")

type = parameters.get("room-type")

indate = str(parameters.get("in-date"))[:10]

outdate = str(parameters.get("out-date"))[:10]

availtypes={"S":"studio","1":"1-bedroom","2":"2-bedroom","3":"3-bedroom", "4":"4-bedroom"}

bedrooms = str(availtypes[type])

query = str(zone)

geolocator = Nominatim(user\_agent="ow\_faq\_bot")

location = geolocator.geocode(query)

print(location.address)

#print(location.raw)

lat, lng = location.latitude, location.longitude

print(lat, lng)

baseurl = 'https://www.oakwood.com/search?city='+query.title()+'&lat='+ str(lat) + '&lng=' + str(lng) + '&moveIn='+ indate + '&moveOut=' + outdate + '&rooms=' + bedrooms

returnstring = 'Please follow the link below to view availability near '+ zone.title() + ': ' + baseurl

#returnstring += '\nThe fastest way to get pricing and booking information is by giving us a call at 1-866-400-1925.\nIs there anything else I can help you with?'

elif(action =="checkavail"):

parameters = result.get("parameters")

zone = parameters.get("room-location")

if(zone.lower() == "warszawa"):

zone = "warsaw"

type = parameters.get("room-type")

indate = str(parameters.get("in-date"))[:10]

outdate = str(parameters.get("out-date"))[:10]

availtypes={"S":"studio","1":"1-bedroom","2":"2-bedroom","3":"3-bedroom", "4":"4-bedroom"}

bedrooms = str(availtypes[type])

query = str(zone)

geolocator = Nominatim(user\_agent="ow\_faq\_bot")

location = geolocator.geocode(query)

print(location.address)

#print(location.raw)

lat, lng = location.latitude, location.longitude

print(lat, lng)

baseurl = 'https://www.oakwood.com/search?city='+query.title()+'&lat='+ str(lat) + '&lng=' + str(lng) + '&moveIn='+ indate + '&moveOut=' + outdate + '&rooms=' + bedrooms

returnstring = 'Please follow the link below to view availability near '+ zone.title() + ': ' + baseurl

#returnstring += '\nThe fastest way to get pricing and booking information is by giving us a call at 1-866-400-1925.\nIs there anything else I can help you with?'

if(returnstring==""):

returnstring="Sorry, I could not find any pricing for that search request. Please try again with a different city. "

print("Response:")

res = {

"fulfillmentText": returnstring,

"source": "Pricing"

}

res = json.dumps(res, indent=4)

print(res)

r = make\_response(res)

r.headers['Content-Type'] = 'application/json'

return r

def fuzzysearch(splitcr,zone,month,type,types):

i = 0

fuzzyratio = 0

partialfuzzyratio=0

foundfuzzy = False

foundzone = ""

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8').replace('"', '')

checkarray=check.split(',')

market = checkarray[0].lower()

if(fuzz.partial\_ratio(zone.lower(), market)>57 and fuzz.partial\_ratio(zone.lower(), market)>partialfuzzyratio and fuzz.ratio(zone.lower(), market)>fuzzyratio):

partialfuzzyratio =fuzz.partial\_ratio(zone.lower(), market)

fuzzyratio =fuzz.ratio(zone.lower(), market)

foundfuzzy=True

foundzone = market

except:

print('weird character')

return foundzone,foundfuzzy

def peakmonthnew(splitcr,zone,month,type,types,currency):

i = 0

peakdict = {"01":[0.0,0.0],"02":[0.0,0.0],"03":[0.0,0.0],"04":[0.0,0.0],"05":[0.0,0.0],"06":[0.0,0.0],"07":[0.0,0.0],"08":[0.0,0.0],"09":[0.0,0.0],"10":[0.0,0.0],"11":[0.0,0.0],"12":[0.0,0.0]}

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8').replace('"', '')

if(zone.lower() in check.lower() and type+"F" in check):

checkarray=check.split(',')

yrmonth=checkarray[1][-2:]

requestct = int(checkarray[3])

totqtrate = float(checkarray[4])

peakdict[yrmonth][0]+=requestct

peakdict[yrmonth][1]+=totqtrate

#avgrate,numberrooms,month

peakarray = ["0.0","0.0","N\A"]

for k in peakdict.keys():

if(peakdict[k][0]!=0):

avgrate = str(round(float(peakdict[k][1]/peakdict[k][0]),2))

if(float(peakarray[0])<float(avgrate)):

peakarray[1]=peakdict[k][0]

peakarray[0]=avgrate

peakarray[2]=k

except:

print('error finding the peak season')

if(peakarray[0]!="0.00" and peakarray[1]!="0.0" and (currency=="" or currency=="USD")):

returnstring="The peak month in "+ zone.title() + " for a "+ types[type]+ " is $"+ peakarray[0] + " in " + calendar.month\_name[int(peakarray[2])] + " based off " + str(int(peakarray[1])) + " rented apartment(s)"

elif(peakarray[0]!="0.00" and peakarray[1]!="0.0"):

c = CurrencyRates()

converted=c.convert('USD', currency, float(peakarray[0]))

returnstring="The peak month in "+ zone.title() + " for a "+ types[type]+ " is "+ str(round(converted,2))+" "+currency+ " in " + calendar.month\_name[int(peakarray[2])] + " based off " + str(int(peakarray[1])) + " rented apartment(s)"

else:

returnstring=""

return returnstring

def peakseason(splitcr,zone,month,type,types,currency):

i = 0

peakdict = {"Summer":[0.0,0.0],"Spring":[0.0,0.0],"Fall":[0.0,0.0],"Winter":[0.0,0.0]}

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8').replace('"', '')

if(zone.lower() in check.lower() and type+"F" in check):

checkarray=check.split(',')

yrmonth=checkarray[1][-2:]

requestct = int(checkarray[3])

totqtrate = float(checkarray[4])

if(yrmonth == "06" or yrmonth == "07" or yrmonth == "08"):

peakdict["Summer"][0]+=requestct

peakdict["Summer"][1]+=totqtrate

elif(yrmonth == "09" or yrmonth == "10" or yrmonth == "11"):

peakdict["Fall"][0]+=requestct

peakdict["Fall"][1]+=totqtrate

elif(yrmonth == "12" or yrmonth == "01" or yrmonth == "02"):

peakdict["Winter"][0]+=requestct

peakdict["Winter"][1]+=totqtrate

elif(yrmonth == "03" or yrmonth == "04" or yrmonth == "05"):

peakdict["Spring"][0]+=requestct

peakdict["Spring"][1]+=totqtrate

peakarray = ["0.0","N\A"]

for k in peakdict.keys():

avgrate = str(round(float(peakdict[k][1]/peakdict[k][0]),2))

if(float(peakarray[0])<float(avgrate)):

maxrooms=peakdict[k][0]

peakarray[0]=avgrate

peakarray[1]=k

market = checkarray[0]

roomtype = checkarray[2]

except:

print('error finding the peak season')

if(peakarray[0]!="0.00" and peakarray[1]!="N\A"and (currency=="" or currency=="USD")):

returnstring="The peak season in "+ zone.title() + " for a "+ types[type]+ " is $"+ peakarray[0] + " in the " + peakarray[1] + " based off " + str(int(maxrooms)) + " rented apartment(s)"

elif(peakarray[0]!="0.00" and peakarray[1]!="N\A"):

c = CurrencyRates()

converted=c.convert('USD', currency, float(peakarray[0]))

returnstring="The peak season in "+ zone.title() + " for a "+ types[type]+ " is "+ str(round(converted,2))+" "+currency+ " in the " + peakarray[1] + " based off " + str(int(maxrooms)) + " rented apartment(s)"

else:

returnstring=""

return returnstring

def lowmonthnew(splitcr,zone,month,type,types,currency):

i = 0

peakdict = {"01":[0.0,0.0],"02":[0.0,0.0],"03":[0.0,0.0],"04":[0.0,0.0],"05":[0.0,0.0],"06":[0.0,0.0],"07":[0.0,0.0],"08":[0.0,0.0],"09":[0.0,0.0],"10":[0.0,0.0],"11":[0.0,0.0],"12":[0.0,0.0]}

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8').replace('"', '')

if(zone.lower() in check.lower() and type+"F" in check):

checkarray=check.split(',')

yrmonth=checkarray[1][-2:]

requestct = int(checkarray[3])

totqtrate = float(checkarray[4])

peakdict[yrmonth][0]+=requestct

peakdict[yrmonth][1]+=totqtrate

#avgrate,numberrooms,month

peakarray = ["10000000000000.0","0.0","N\A"]

for k in peakdict.keys():

if(peakdict[k][0]!=0):

avgrate = str(round(float(peakdict[k][1]/peakdict[k][0]),2))

if(float(peakarray[0])>float(avgrate)):

peakarray[1]=peakdict[k][0]

peakarray[0]=avgrate

peakarray[2]=k

except:

print('error finding the peak season')

if(peakarray[0]!="10000000000000.0" and peakarray[1]!="0.0"and (currency=="" or currency=="USD")):

returnstring="The cheapest month in "+ zone.title() + " for a "+ types[type]+ " is $"+ peakarray[0] + " in " + calendar.month\_name[int(peakarray[2])] + " based off " + str(int(peakarray[1])) + " rented apartment(s)"

elif(peakarray[0]!="10000000000000.0" and peakarray[1]!="0.0"):

c = CurrencyRates()

converted=c.convert('USD', currency, float(peakarray[0]))

returnstring="The cheapest month in "+ zone.title() + " for a "+ types[type]+ " is "+ str(round(converted,2))+" "+currency+ " in " + calendar.month\_name[int(peakarray[2])] + " based off " + str(int(peakarray[1])) + " rented apartment(s)"

else:

returnstring=""

return returnstring

def lowseason(splitcr,zone,month,type,types,currency):

i = 0

peakdict = {"Summer":[0.0,0.0],"Spring":[0.0,0.0],"Fall":[0.0,0.0],"Winter":[0.0,0.0]}

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8').replace('"', '')

if(zone.lower() in check.lower() and type+"F" in check):

checkarray=check.split(',')

yrmonth=checkarray[1][-2:]

requestct = int(checkarray[3])

totqtrate = float(checkarray[4])

if(yrmonth == "06" or yrmonth == "07" or yrmonth == "08"):

peakdict["Summer"][0]+=requestct

peakdict["Summer"][1]+=totqtrate

elif(yrmonth == "09" or yrmonth == "10" or yrmonth == "11"):

peakdict["Fall"][0]+=requestct

peakdict["Fall"][1]+=totqtrate

elif(yrmonth == "12" or yrmonth == "01" or yrmonth == "02"):

peakdict["Winter"][0]+=requestct

peakdict["Winter"][1]+=totqtrate

elif(yrmonth == "03" or yrmonth == "04" or yrmonth == "05"):

peakdict["Spring"][0]+=requestct

peakdict["Spring"][1]+=totqtrate

peakarray = ["10000000000000.0","N\A"]

for k in peakdict.keys():

avgrate = str(round(float(peakdict[k][1]/peakdict[k][0]),2))

if(float(peakarray[0])>float(avgrate)):

maxrooms=peakdict[k][0]

peakarray[0]=avgrate

peakarray[1]=k

market = checkarray[0]

roomtype = checkarray[2]

except:

print('error finding the low season')

if(peakarray[0]!="0.00" and peakarray[1]!="N\A"and (currency=="" or currency=="USD")):

returnstring="The cheapest season in "+ zone.title() + " for a "+ types[type]+ " is $"+ peakarray[0] + " in the " + peakarray[1] + " based off " + str(int(maxrooms)) + " rented apartment(s)"

elif(peakarray[0]!="0.00" and peakarray[1]!="N\A"):

c = CurrencyRates()

converted=c.convert('USD', currency, float(peakarray[0]))

returnstring="The cheapest season in "+ zone.title() + " for a "+ types[type]+ " is "+ str(round(converted,2))+" "+currency+ " in the " + peakarray[1] + " based off " + str(int(maxrooms)) + " rented apartment(s)"

else:

returnstring=""

return returnstring

def summer(splitcr,zone,month,type,types,currency):

i = 0

found = False

requestct = 0

totqtrate = 0

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8').replace('"', '')

if(zone.lower() in check.lower() and type+"F" in check):

checkarray=check.split(',')

yearmonth = checkarray[1][-2:]

if("06" == yearmonth or "07" == yearmonth or "08" == yearmonth):

roomtype = checkarray[2]

market = checkarray[0]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

found=True

except:

print('weird character')

if(found == True and (currency=="" or currency=="USD")):

returnstring="The average price is $"+ str(round(float(totqtrate/requestct),2))+ " for a "+ types[roomtype[0]]+" in "+zone.title() + " during the " + month.capitalize() + " based off " + str(requestct) + " rented apartment(s)"

elif(found == True):

c = CurrencyRates()

converted=c.convert('USD', currency, float(totqtrate/requestct))

returnstring="The average price is "+ str(round(converted,2))+" "+currency+ " for a "+ types[roomtype[0]]+" in "+zone.title() + " during the " + month.capitalize() + " based off " + str(requestct) + " rented apartment(s)"

else:

returnstring=""

return returnstring,found

def spring(splitcr,zone,month,type,types,currency):

i = 0

found = False

requestct = 0

totqtrate = 0

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8').replace('"', '')

if(zone.lower() in check.lower() and type+"F" in check):

checkarray=check.split(',')

yearmonth = checkarray[1][-2:]

if("03" == yearmonth or "04" == yearmonth or "05" == yearmonth):

market = checkarray[0]

roomtype = checkarray[2]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

found=True

except:

print('weird character')

if(found == True and (currency=="" or currency=="USD")):

returnstring="The average price is $"+ str(round(float(totqtrate/requestct),2))+ " for a "+ types[roomtype[0]]+" in "+zone.title() + " during the " + month.capitalize() + " based off " + str(requestct) + " rented apartment(s)"

elif(found == True):

c = CurrencyRates()

converted=c.convert('USD', currency, float(totqtrate/requestct))

returnstring="The average price is "+ str(round(converted,2))+" "+currency+ " for a "+ types[roomtype[0]]+" in "+zone.title() + " during the " + month.capitalize() + " based off " + str(requestct) + " rented apartment(s)"

else:

returnstring=""

return returnstring,found

def fall(splitcr,zone,month,type,types,currency):

i = 0

found = False

requestct = 0

totqtrate = 0

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8').replace('"', '')

if(zone.lower() in check.lower() and type+"F" in check):

checkarray=check.split(',')

yearmonth = checkarray[1][-2:]

if("09" == yearmonth or "10" == yearmonth or "11" == yearmonth):

market = checkarray[0]

roomtype = checkarray[2]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

found=True

except:

print('weird character')

if(found == True and (currency=="" or currency=="USD")):

returnstring="The average price is $"+ str(round(float(totqtrate/requestct),2))+ " for a "+ types[roomtype[0]]+" in "+zone.title() + " during the " + month.capitalize() + " based off " + str(requestct) + " rented apartment(s)"

elif(found == True):

c = CurrencyRates()

converted=c.convert('USD', currency, float(totqtrate/requestct))

returnstring="The average price is "+ str(round(converted,2))+" "+currency+ " for a "+ types[roomtype[0]]+" in "+zone.title() + " during the " + month.capitalize() + " based off " + str(requestct) + " rented apartment(s)"

else:

returnstring=""

return returnstring,found

def winter(splitcr,zone,month,type,types,currency):

i = 0

found = False

requestct = 0

totqtrate = 0

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8').replace('"', '')

if(zone.lower() in check.lower() and type+"F" in check):

checkarray=check.split(',')

yearmonth = checkarray[1][-2:]

if("01" == yearmonth or "02" == yearmonth or "12" == yearmonth):

market = checkarray[0]

roomtype = checkarray[2]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

found=True

except:

print('weird character')

if(found == True and (currency=="" or currency=="USD")):

returnstring="The average price is $"+ str(round(float(totqtrate/requestct),2))+ " for a "+ types[roomtype[0]]+" in "+zone.title() + " during the " + month.capitalize() + " based off " + str(requestct) + " rented apartment(s)"

elif(found == True):

c = CurrencyRates()

converted=c.convert('USD', currency, float(totqtrate/requestct))

returnstring="The average price is "+ str(round(converted,2))+" "+currency+ " for a "+ types[roomtype[0]]+" in "+zone.title() + " during the " + month.capitalize() + " based off " + str(requestct) + " rented apartment(s)"

else:

returnstring=""

return returnstring,found

def monthly(splitcr,zone,month,type,types,currency):

i = 0

found = False

requestct = 0

totqtrate = 0

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8').replace('"', '')

if(zone.lower() in check.lower() and type+"F" in check):

print("FOUND: ",check)

checkarray=check.split(',')

yearmonth = checkarray[1][-2:]

if(month == yearmonth):

market = checkarray[0]

roomtype = checkarray[2]

avgrate = checkarray[6]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

found=True

except:

#check = splitcr[i].decode('utf-8').replace('"', '')

print("weird char")

#print('check: ',check)

if(found == True and (currency=="" or currency=="USD")):

returnstring="The average price is $"+ str(round(float(totqtrate/requestct),2))+ " for a "+ types[roomtype[0]]+" in "+zone.title() + " during " + calendar.month\_name[int(month)] + " based off " + str(int(requestct)) + " rented apartment(s)"

elif(found == True):

c = CurrencyRates()

converted=c.convert('USD', currency, float(totqtrate/requestct))

returnstring="The average price is "+ str(round(converted,2))+" "+currency+ " for a "+ types[roomtype[0]]+" in "+zone.title() + " during "+ calendar.month\_name[int(month)] + " based off " + str(int(requestct)) + " rented apartment(s)"

else:

returnstring=""

return returnstring,found

def yearly(splitcr,zone,month,type,types,currency):

i = 0

found = False

requestct = 0

totqtrate = 0

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8').replace('"', '')

if(zone.lower() in check.lower() and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

avgrate = checkarray[6]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

found=True

except:

print('weird character')

if(found == True and (currency=="" or currency=="USD")):

returnstring="I could not find monthly data for that request. However, the average yearly rate is $"+ str(round(float(totqtrate/requestct),2))+ " for a "+ types[roomtype[0]]+ " in "+ zone.title() + " based off " + str(requestct) + " rented apartment(s)"

elif(found == True):

c = CurrencyRates()

converted=c.convert('USD', currency, float(totqtrate/requestct))

returnstring="I could not find monthly data for that request. However, the average yearly rate is "+ str(round(converted,2))+" "+currency+ " for a "+ types[roomtype[0]]+" in "+zone.title() +" based off " + str(int(requestct)) + " rented apartment(s)"

else:

returnstring=""

return returnstring,found

def yearlylast(splitcr,zone,month,type,types,currency):

i = 0

found = False

requestct = 0

totqtrate = 0

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8').replace('"', '')

if(zone.lower() in check.lower()):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

avgrate = checkarray[6]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

found=True

except:

print('weird character')

if(found == True and (currency=="" or currency=="USD")):

returnstring="I could not find monthly data for that room type. However, the average yearly rate is $"+ str(round(float(totqtrate/requestct),2))+ " for all room types in "+ zone.title() + " based off " + str(requestct) + " rented apartment(s)"

elif(found == True):

c = CurrencyRates()

converted=c.convert('USD', currency, float(totqtrate/requestct))

returnstring="I could not find monthly data for that room type. However, the average yearly rate is "+ str(round(converted,2))+" "+currency+ " for all room types in "+zone.title() +" based off " + str(int(requestct)) + " rented apartment(s)"

else:

returnstring=""

return returnstring,found

def year(splitcr,zone,month,type,types,currency):

i = 0

found = False

requestct = 0

totqtrate = 0

while i<len(splitcr)-1:

i+=1

try:

check = splitcr[i].decode('utf-8').replace('"', '')

if(zone.lower() in check.lower() and type+"F" in check):

checkarray=check.split(',')

market = checkarray[0]

roomtype = checkarray[2]

avgrate = checkarray[6]

requestct += int(checkarray[3])

totqtrate += float(checkarray[4])

found=True

except:

print('weird character')

if(found == True and (currency=="" or currency=="USD")):

returnstring="The average yearly rate is $"+ str(round(float(totqtrate/requestct),2))+ " for a "+ types[roomtype[0]]+ " in "+ zone.title() + " based off " + str(requestct) + " rented apartment(s)"

elif(found == True):

c = CurrencyRates()

converted=c.convert('USD', currency, float(totqtrate/requestct))

returnstring="The average yearly rate is "+ str(round(converted,2))+" "+currency+ " for a "+ types[roomtype[0]]+" in "+zone.title() +" based off " + str(int(requestct)) + " rented apartment(s)"

else:

returnstring=""

return returnstring,found

def getCSV():

with urlopen("https://storage.googleapis.com/pricing-bot-production.appspot.com/TopCityNoComma.csv") as url:

cr = url.read()

splitcr = cr.splitlines()

return splitcr

if \_\_name\_\_ == '\_\_main\_\_':

port = int(os.getenv('PORT', 8080))

print ("Starting app on port %d" %(port))

app.run(debug=True, port=port, host='0.0.0.0')

**App.yaml**

runtime: python

env: flex

entrypoint: gunicorn -b :$PORT app:app

runtime\_config:

python\_version: 3

manual\_scaling:

instances: 1

resources:

cpu: 1

memory\_gb: 0.5

disk\_size\_gb: 10

**Requirements.txt**

Flask==0.12.2

google-api-python-client==1.6.6

gunicorn==19.7.1

fuzzywuzzy==0.17.0

forex-python==1.1

geopy==1.16.0