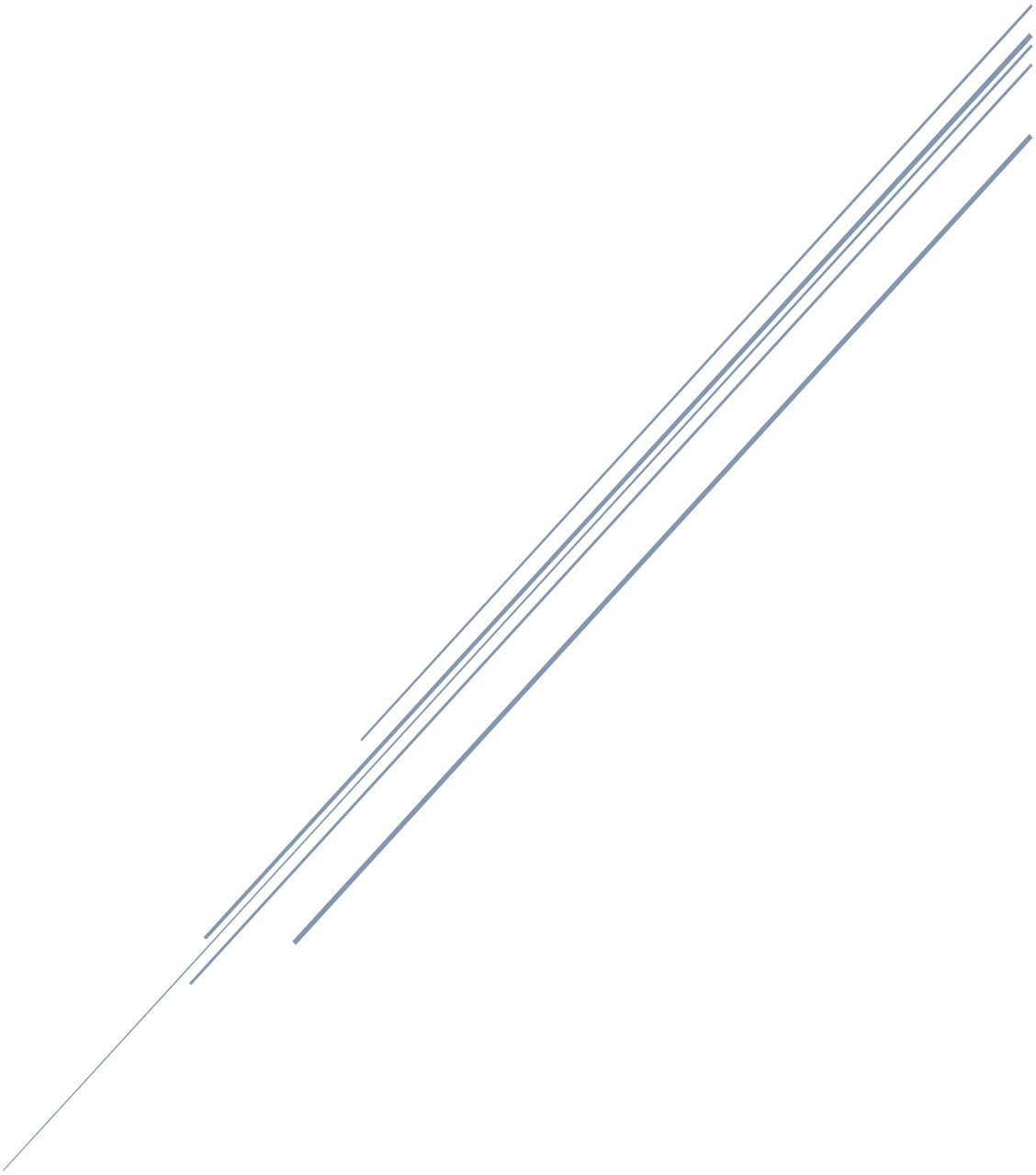
WEBHOOK DOCUMENTATION

FIRST LIFE



First Life Webhook Documentation – Google Assistant

Abbiamo utilizzato **node js** e **npm** per installare il pacchetto **action-on-google** per gestire gli intenti del chatbot che li abbiamo definiti tramite dialogflow e gestire tutte le comunicazioni tra l'utente e il chatbot in questo codice webhook.

Abbiamo quattro file Javascript all'interno di questo codice che insieme gestiscono la conversazione tra l'utente e il chatbot, gestiscono gli intenti e i parametri del dialogflow e infine creano la risposta del chatbot in base alla richiesta dell'utente e ai parametri della conversazione.

- **constant.js**
  Include parti costanti del codice.

- **services.js**
  Gestisce i dati e le richieste al server per ottenere i dati.

- **functions.js**
  include tutte le funzioni e le logiche per trovare la posizione dell'utente o punto vendita o ristorante o azienda utilizzando l'API di Google Map e creare la risposta finale del chatbot.

- **index.js**
  gestisce gli intenti e i parametri del flusso di dialogo che provengono dalla conversazione tra l'utente e il chatbot e mostra la risposta finale all'utente in base alla richiesta dell'utente.

Discuteremo di ciascuno separatamente in questa documentazione.

Anche il codice stesso è completamente documentato all'interno.

constants.js

Qui creiamo la prima parte e l'ultima parte della risposta del chatbot in base ai parametri della conversazione tra chatbot e utente, che sono:

1-posizione

2-attivita

Ad esempio, se l'utente chiede al chatbot: "*dove posso mangiare ricotta nella mia citta*"

attivita = mangiare

posizione = citta

FUNCTIONS:

function primaParte(attivita, posizione)

Crea la prima parte della risposta utilizzando **attività** e **posizione**:

```
/**
 * crea la parte principale della risposta del chatbot in base ai parametri della conversazione tra chatbot e l'utente
 * @param attivita --- potrebbe essere 'mangiare' o 'comprare' o... e viene da parametri di conversazione
 * @param posizione --- potrebbe essere 'dintorni' o 'citta' o 'regione' o... e viene da parametri di conversazione
 * @returns {string}
 */
function primaParte(attivita, posizione){
    let primaParte = '';
    switch (posizione){
        case 'dintorni':
            primaParte += `Nel raggio di 500 mt dalla tua posizione, `;
            if (attivita == 'comprare' || attivita == 'trovare' || attivita == 'acquistare' || attivita == 'mangiare') {
                primaParte += `puoi ${attivita}, `
            } else if (attivita == 'producono' || attivita == 'fanno' || attivita == 'confezionano' || attivita == 'vengono prodotti'
                primaParte += `${attivita}, `
            }
            break;
        case 'citta':
            primaParte += `Nella tua citta, `;
            if (attivita == 'comprare' || attivita == 'trovare' || attivita == 'acquistare' || attivita == 'mangiare') {
                primaParte += `puoi ${attivita}, `
            } else if (attivita == 'producono' || attivita == 'fanno' || attivita == 'confezionano' || attivita == 'vengono prodotti'
                primaParte += `${attivita}, `
            }
            break;
```

Ad esempio, se l'utente chiede al chatbot: "*dove posso mangiare ricotta nella mia regione*",

posizione = regione

attivita = mangiare

Quindi, la prima parte della risposta è:

"*nella tua città puoi mangiare*"

Abbiamo "*nella tua città*" perché **posizione** che è uguale a **citta**, e abbiamo "*puoi mangiare*", perché **attività** che è uguale a **mangiare**.

Quindi possiamo manipolare questa parte come vogliamo facilmente.

function altro()

Crea l'ultima parte della risposta:

```
/**
 * crea la parte finale della risposta
 * @returns {string}
 */
function altro(){
    return ". vuoi sapere altro?";
}
```

function fallBack()

Crea la risposta quando il chatbot non è in grado di capire cosa sta dicendo l'utente.

```
/**
 * crea una risposta di fallback
 * @returns {string}
 */
function fallBack(){
    return `non ho capito bene, puoi ripetere?`;
}
```

Possiamo cambiare ogni parte separatamente come vogliamo, e non c'è bisogno di cambiare un'altra parte del codice se vogliamo modificare la risposta del chatbot.

function creaCodice(categoria);

crea una corrispondenza per qualsiasi categoria, tra la categoria che otteniamo dall'**Entity** di dialogflow e la categoria reale che abbiamo nei nostri server.

Ad esempio, quando l'utente chiede a chatbot, "*quali formaggi posso trovare*?", la categoria che dialogflow ci dà è **formaggi** ma non possiamo inviare una richiesta al server utilizzando **formaggi**, perché forse la vera categoria che abbiamo nel server è **FORMAGGI**. Quindi, la funzione **creaCodice** risolve questo problema per noi e crea una corrispondenza

```
/**
 * crea un codice unico per ogni categoria
 * @param categoria
 * @returns {string}
 */
function creaCodice(categoria){
    let codice = '';
    switch (categoria){
        case 'formaggi':
            codice = 'FORMAGGI';
            break;
        case 'ricotte':
            codice = 'RICOTTE';
            break;
        case 'mozzarelle':
            codice = 'MOZZARELLE';
            break;
```

Questa funzione restituisce **codice**.

Possiamo aggiungere categorie e creare una corrispondenza tra quella che otteniamo da dialogflow e quella che abbiamo nel server.

Attenzione: dobbiamo cambiare le partite quando abbiamo accesso a dati reali

function raggio()

restituisce il raggio che vogliamo cercare intorno all'utente per trovare **punti vendita** o **ristoranti** o **aziende**.

Lo usiamo solo quando l'utente chiede al chatbot di cercare intorno a lui. (es. quando l'utente menziona "*nei dintorni*" o "*vicino a me*")

```
/**
 * il raggio intorno all'utente per il calcolo della distanza e controlla se c'è qualche 'ristorante' o 'azienda' o
 * @returns {number}
 */
function raggio(){
    return 1000;
}
```

services.js

include funzioni per ottenere i dati dal server.

Utilizziamo parametri di conversazione tra il chatbot e l'utente e inviamo una richiesta al server e otteniamo i dati.

Il parametro che utilizziamo per inviare la richiesta è **categoria** che viene dalla conversazione tra utente e chatbot. Ad esempio, quando l'utente dice “dove posso acquistare taralli nei dintorni?”

categoria = taralli

ma come abbiamo detto prima **taralli** è la categoria che otteniamo da dialogflow e non quella che abbiamo nel nostro server, quindi non possiamo inviare richiesta con **taralli**, ma abbiamo risolto questo problema prima creando la funzione **creaCodice** in costants.js. e come vedremo in seguito per accedere ai dati sul server utilizziamo prima la funzione **creaCodice** per fare una corrispondenza tra ciò che abbiamo ottenuto da dialogflow e ciò che abbiamo nel nostro server e poi creiamo i nostri dati fittizi.

FUNCTIONS

function findProdotti(categoria)

questa è la funzione principale e dobbiamo modificare questa funzione e ottenere i dati dal server usando rest. In questo momento abbiamo un mock-data che include due singoli prodotti per ogni categoria.

Ad esempio, per la categoria **FORMAGGI** abbiamo un singolo prodotto con codice **FORMAGGIO_1** e un altro con codice **FORMAGGIO_2**.

Ogni singolo prodotto ha le sue proprietà come **descrizione**, **categoria**, **aziende**, **punti_vendita** e **ristoranti**.

punti_vendita, **ristorante**, e **aziende** sono oggetti, quindi ad esempio possiamo avere più di una ristoranti per un singolo prodotto.

```
function findProdotti(categoria) {
    let prodotti = null;
    if (constants.creaCodice(categoria) == "FORMAGGI") {
        prodotti = [{
            codice: "FORMAGGIO_1",
            descrizione: "Fiore Sardo",
            categoria: "FORMAGGI",
            aziende: findAziende( categoria: "FORMAGGI")[0],
            punti_vendita: findPuntiVendita( categoria: "FORMAGGI")[0],
            ristoranti: findRistoranti( categoria: "FORMAGGI")[0]
        }, {
            codice: "FORMAGGIO_2",
            descrizione: "Asiago",
            categoria: "FORMAGGI",
            aziende: findAziende( categoria: "FORMAGGI")[0],
            punti_vendita: findPuntiVendita( categoria: "FORMAGGI")[0],
            ristoranti: findRistoranti( categoria: "FORMAGGI")[0]
        }];
```

Attenzione: usiamo prima **creaCodice** da constants.js.

function findPuntiVendita(categoria)

comprende dei mock punti vendita per ogni categoria.

```
function findPuntiVendita(categoria) {
    let puntiVendita = null;
    if(categoria == 'FORMAGGI'){
        puntiVendita = [{
            codice: "PUNTO_VENDITA_1",
            descrizione: "Punto vendita 1",
            indirizzo: "Corso Torino, 118, 10056 Oulx TO",
            contatto_telefonico: "+390122833800",
            posizione: {lat: 45.040350, lng: 6.842267}
        }];
    } else if(categoria == 'RICOTTE'){
        puntiVendita = [{
            codice: "PUNTO_VENDITA_2",
            descrizione: "Punto vendita 2",
            indirizzo: "Corso Torino, 118, 10056 Oulx TO",
            contatto_telefonico: "+390122833800",
            posizione: {lat: 45.040350, lng: 6.842267}
        }];
```

ogni punto vendita ha le sue proprietà come **codice**, **descrizione**, **indirizzo**, **contatto_telefonico** e **posizione**. Usiamo **posizione** in functions.js per trovare la posizione precisa del punto vendita.

function findRistoranti(categoria)

include qualche mock ristorante per ogni categoria.

```
function findRistoranti(categoria) {
    let ristoranti = null;
    if(categoria == 'FORMAGGI'){
        ristoranti = [{
            codice: "RISTORANTE_1",
            descrizione: "Ristorante 1",
            indirizzo: "Corso Torino, 118, 10056 Oulx TO",
            contatto_telefonico: "+390122833800",
            posizione: {lat: 45.040350, lng: 6.842267}
        }];
    } else if(categoria == 'RICOTTE'){
        ristoranti = [{
            codice: "RISTORANTE_2",
            descrizione: "Ristorante 2",
            indirizzo: "Corso Torino, 118, 10056 Oulx TO",
            contatto_telefonico: "+390122833800",
            posizione: {lat: 45.040350, lng: 6.842267}
        }];
```

ogni ristorante ha le sue proprietà come **codice**, **descrizione**, **indirizzo**, **contatto_telefonico** e **posizione**. Usiamo **posizione** in functions.js per trovare la posizione precisa del ristorante.

function findAziende(categoria)

include alcuni mock azienda per ogni categoria.

```
function findAziende(categoria) {
    let aziende = null;
    if(categoria == 'FORMAGGI'){
        aziende = [{
            codice: "AZIENDA_1",
            descrizione: "Azienda 1",
            indirizzo: "Fraz. San Sicario, 1 Cesana Torinese (TO)",
            contatto_telefonico: "+390124356982",
            posizione: {lat: 44.966535, lng: 6.817225}
        }];
    } else if(categoria == 'RICOTTE'){
        aziende = [{
            codice: "AZIENDA_2",
            descrizione: "Azienda 2",
            indirizzo: "Fraz. San Sicario, 1 Cesana Torinese (TO)",
            contatto_telefonico: "+390124356982",
            posizione: {lat: 44.966535, lng: 6.817225}
        }];
```

ogni azienda ha le sue proprietà come **codice**, **descrizione**, **indirizzo**, **contatto_telefonico** e **posizione**. Usiamo **posizione** in functions.js per trovare la posizione precisa dell'azienda.

Attenzione: per ora i dati che abbiamo, sono fittizi e questi dati non sono reali e non provengono dal nostro server. Per ottenere dati reali dobbiamo inviare la richiesta al server utilizzando rest.

Quindi, dobbiamo cambiare completamente la funzione **findProdotti**. E dopo, non abbiamo più bisogno di **findPuntiVendita** e **findRistoranti** e **findAziende** e probabilmente possiamo eliminarli.

functions.js

include tutte le logiche e la funzione che crea la risposta finale del chatbot.

Dobbiamo modificare le funzioni qui quando abbiamo accesso a dati reali e anche all'API di Google Map.

Abbiamo un oggetto qui **_posizione_utente** che mantiene la posizione dell'utente che otteniamo dall'account Google dell'utente. Possiamo ottenere questi parametri: **città**, **indirizzo formattato**, **codice postale**, **coordinate** (come oggetto).

```
/**
 * - possiamo ottenere le 'coordinate' o 'city' o 'formattedAddress' o 'zipCode' dell'utente da '_posizione_utente'.
 * - viene dall'account Google dell'utente.
 * @type object
 * @private
 */
let _posizione_utente = {};
```

FUNCTIONS:

function findDistanza(posizione_prodotto)

posizione_prodotto : posizione dei **punti vendita**, **ristoranti** o **aziende** del prodotto specifico, in base alla richiesta dell'utente e ai parametri di conversazione.

```
function findDistanza(posizione_prodotto) {
    /*
     * let lng_prodotto = posizione_prodotto.lng;
     * let lat_prodotto = posizione_prodotto.lat;
     *** così possiamo trovare la posizione di 'ristorante' o 'punti_vendita' o 'azienda', utilizzando Google Map API
     *** abbiamo anche le coordinate dell'utente (viene da '_posizione_utente'), quindi possiamo trovare anche la posiz
     *** quindi possiamo trovare la distanza tra l'utente e "punto vendita" o "ristorante" o "azienda" di "prodotto", u
     * if(d < raggio){
     *     return true,
     * } else{
     *     return false
     * }
     *** per ora questa funzione restituisce sempre "true", perché non abbiamo accesso all'API di Google Map
     */
    console.log("Product Position: ");
    console.log(posizione_prodotto);
    console.log("User Position: ");
    console.log(_posizione_utente.coordinates);
    console.log(_posizione_utente.zipCode);
    return true;
}
```

quando l'utente chiede al chatbot di cercare un prodotto intorno a sé e menziona "*nei dintorni*" o "*intorno a me*" o …, ad esempio quando l'utente dice "*dove posso comprare formaggi nei dintorni*", i parametri di conversazione del chatbot sono:

domanda : dove

categoria : formaggi ("FORMAGGI" perché usiamo la funzione creaCodice())

attivita : comprare

posizione : dintorni

come abbiamo detto prima ogni **punto vendita** o **azienda** o **ristorante** ha una proprietà di **posizione** che è un oggetto comprensivo di coordinate. In questo caso l'utente ha chiesto al chatbot di cercare **punti vendita** intorno a lui, quindi passiamo la posizione di ogni **punto vendita** come argomento della funzione **findDistanza(posizione_prodotto)**, quindi questa funzione è in grado di calcolare la posizione effettiva di ogni **punto vendita** utilizzando l'API di Google Map, inoltre abbiamo le coordinate della posizione dell'utente (_posizione_utente) dall'account Google dell'utente quindi possiamo calcolare anche la posizione effettiva dell'utente utilizzando l'API di Google Map. Questa funzione è quindi in grado di calcolare la distanza tra utente e in questo caso **punti vendita** (se l'utente chiede "*dove posso mangiare formaggi nei dintorni*", passiamo posizione di **ristoranti** di ogni prodotto). Possiamo chiamare questa distanza **d**. infine, questa funzione restituisce **true** "**if (d <= raggio)**" (abbiamo già definito **raggio** in constants.js) che significa che questo **punto vendita** è abbastanza vicino all'utente e chatbot può mostrarglielo.

Questa funzione viene chiamata proprio quando il chatbot cerca un prodotto intorno all'utente.

Per ora la funzione **findDistanza** restituisce **true** perché non abbiamo accesso all'API di Google Map per calcolare le posizioni.

function findCitta(posizione_prodotto)

posizione_prodotto : posizione dei **punti vendita**, **ristoranti** o **aziende** del prodotto specifico, in base alla richiesta dell'utente e ai parametri di conversazione.

```
function findCitta(posizione_prodotto) {
    /*
     * let lng_prodotto = posizione_prodotto.lng;
     * let lat_prodotto = posizione_prodotto.lat;
     *** possiamo trovare città di 'ristorante' o 'punti_vendita' o 'azienda' utilizzando Google Map API
     * let citta = ("citta" of "punto vendita" o "ristorante" o "azienda" of prodotto).toUpperCase();
     *** abbiamo anche "città" di utente (_posizione_utente.city).
     * let uCitta = _posizione_utente.city.toUpperCase();
     *** così possiamo confrontare le città.
     * if(citta == uCitta){
     *     return true;
     * } else{
     *     return false;
     * }
     *** per ora questa funzione restituisce sempre "true", perché non abbiamo accesso all'API di Google Map
     */
    console.log("Product Position: ");
    console.log(posizione_prodotto);
    console.log("User Position: ");
    console.log(_posizione_utente.coordinates);
    console.log(_posizione_utente.zipCode);
    return true;
}
```

quando l'utente chiede al chatbot di cercare un prodotto all'interno della sua città, ad esempio quando l'utente dice "*dove posso comprare formaggi nella mia città*", i parametri di conversazione del chatbot sono:

domanda : dove

categoria : formaggi ("FORMAGGI" perché usiamo la funzione creaCodice())

attivita : comprare

posizione : citta

questa funzione calcola la posizione di **punti vendita** o **ristoranti** o **azienda** esattamente come la funzione **findDistanza** quindi, possiamo trovare la **città** di ogni **punto vendita** utilizzando Google Map API, ma in questo caso non è necessario utilizzare le coordinate dell'utente per trovare la città dell'utente perché possiamo avere la città dell'utente direttamente (come abbiamo detto prima possiamo ottenere **città**, **indirizzo formattato**, **coordinate** e **codice postale** dell'utente dal suo account Google). Quindi dobbiamo solo confrontare la città dell'utente che è "**_posizione_utente.city**" e la città di **punti vendita** o **ristoranti** o **aziende** che abbiamo calcolato utilizzando Google Map API, e se queste due città fossero uguali, la funzione **findCitta** restituisce **true** che significa che questo prodotto ha punti vendita all'interno della città dell'utente in modo che il chatbot possa mostrargli questo punto vendita.

Questa funzione viene chiamata quando il chatbot cerca un prodotto specifico all'interno della città dell'utente.

Per ora la funzione **findCitta** restituisce **true** perché non abbiamo accesso all'API di Google Map per calcolare le posizioni.

function findRegione(posizione_prodotto)

posizione_prodotto : posizione dei **punti vendita**, **ristoranti** o **aziende** del prodotto specifico, in base alla richiesta dell'utente e ai parametri di conversazione.

```
function findRegione(posizione_prodotto) {
    /*
     * let lng_prodotto = posizione_prodotto.lng;
     * let lat_prodotto = posizione_prodotto.lat;
     *** quindi possiamo trovare la regione de "punto vendita" o "ristorante" o "azienda" del prodotto utilizzando l'A
     * let regione = ("regione" di "punto vendita" o "ristorante" o "azienda" of prodotto).toUpperCase();
     *** abbiamo anche le "coordinate" dell'utente (_posizione_utente.coordinates).
     * let uRegione = (user's "regione").toUpperCase();
     *** così possiamo confrontare le regioni.
     * if(regione == uRegione){
     *     return true;
     * } else{
     *     return false;
     * }
     *** per ora questa funzione restituisce sempre "true", perché non abbiamo accesso all'API di Google Map
     */
    console.log("Product Position: ");
    console.log(posizione_prodotto);
    console.log("User Position: ");
    console.log(_posizione_utente.coordinates);
    console.log(_posizione_utente.zipCode);
    return true;
}
```

Questa funzione viene chiamata quando l'utente chiede al chatbot di cercare un prodotto specifico all'interno della sua **regione**, e agisce esattamente come la funzione **findCitta**. L'unica differenza è, in **findRegione** non possiamo ottenere la regione dell'utente direttamente dal suo account Google e dobbiamo ottenere la regione utilizzando le coordinate dell'utente e l'API di Google Map. Il resto è lo stesso con la funzione "findCitta".

Per ora questa funzione restituisce anche **true** perché non abbiamo accesso all'API di Google Map.

function risposta(categorie, domanda, attivita, posizione, posizione_utente)

questa funzione crea la risposta finale del chatbot in base ai **parametri** di conversazione tra utente e chatbot e le tre funzioni **findDistanza**, **findCitta** e **findRegione** in base alla richiesta dell'utente.

```
/**
 * crea risposta di chatbot
 * @param categoria --- potrebbe essere "formaggi" o "ricotte" o ... e viene da 'parameters' della conversazione del ch
 * @param domanda --- potrebbe essere "dove (o "in quale posto")" o "quali" e viene da 'parameters' della conversazione
 * @param attivita --- potrebbe essere "mangiare", "comprare", "trovare", ... e viene da 'parameters' della conversazio
 * @param posizione --- potrebbe essere "dintorni ("nei dintorni" o "intorno a me" o ...)", "citta" o "regione" e viene
 * @param posizione_utente --- possiamo ottenerlo dall'account Google dell'utente (solo quando l'utente utilizza la pia
 * @returns {string} --- risposta di chatbot
 */
function risposta(categoria, domanda, attivita, posizione, posizione_utente) {...}
```

Parameters: (vengono da conversazione tra il chatbot e l'utente)

categorie : potrebbe essere "*formaggi*", "*taralli*" or …

domanda : potrebbe essere "*dove*" or "*quali*"

attivita : potrebbe essere "*mangiare*", "*comprare*", "*acquistare*" or …

posizione : potrebbe essere "*dintorni*", "*citta*" or "*regione*"

posizione_utente : questa è la posizione dell'utente che possiamo ottenere dal suo account Google e salvarla come oggetto in **_posizione_utente** (l'abbiamo definita prima in functions.js).

questa funzione crea la parte principale della risposta del chatbot in base a questi parametri e la aggiunge alla prima parte e all'ultima parte della risposta che abbiamo definito prima in **constants.js** (funzione **primaParte** e **altro** in constants.js) e restituisce la risposta finale di chatbot.

index.js

Qui possiamo gestire tutti gli intenti del chatbot che abbiamo già definito in dialogflow.

First we have installed **actions-on-google** using **npm** and imported **dialogflow** e **Permission** from actions-on-google package.

Abbiamo anche installato il pacchetto **express** utilizzando **npm** per eseguire l'app utilizzando un listener e un **port** specifica, nel nostro caso port potrebbe essere fornita da un server host (es. Google Cloud o Heroku) oppure è **3000**.

```
const port = process.env.PORT || 3000;
const services = require('./services');
const functions = require('./functions');
const express = require('express');
const bodyParser = require('body-parser');
const {
    dialogflow,
    Permission
} = require('actions-on-google');
```

Poi iniziamo a manipolare e gestire ciò che vogliamo che accada in ogni intento e tutte le comunicazioni tra chatbot e l'utente in ogni intento e poi forniamo una risposta adeguata in base alla richiesta dell'utente all'interno di ogni intento.

INTENT HANDLERS

Default Welcome Intent

Gestisce il massaggio di benvenuto e tutto ciò che l'utente vedrà quando inizierà a utilizzare il chatbot.

```
// gestisce il massaggio di benvenuto e tutto ciò che l'utente vedrà quando inizierà a utilizzare il chatbot.
app.intent('Default Welcome Intent', conv => {
    console.log(conv.body.originalDetectIntentRequest.source);
    conv.ask( responses: 'Benvenuto, sono il tuo assistente personale. Posso consigliarti dove mangiare e dove comprare
});
```

Default Welcome Intent è il nome dell'intento in dialogflow e **conv** è un argomento che gestisce la conversazione tra l'utente e il chatbot all'interno di ogni intent. Possiamo ottenere tutti i parametri della conversazione da conv e inoltre possiamo rispondere all'utente con i metodi, **conv.ask("***response***")** e **conv.close("***response***")**, conv.close(), chiude la conversazione dopo aver mostrato la risposta all'utente.

main

quando l'utente chiede qualcosa al chatbot questo intento viene attivato e controlla la richiesta dell'utente e poi gestisce la richiesta dell'utente. se l'utente non specifica una posizione per la ricerca (es. *'nei dintorni'* o *'nella mi città'*), questo intent (main) risponde all'utente utilizzando the function **risposta** which we have already defined in **functions.js** and imported in index.js. In this case the final response is:

risposta(params.categorie, params.domanda, params.attivita, " ", null), viene da functions.js:

```
function risposta(categoria, domanda, attivita, posizione, posizione_utente)
```

passiamo una **stringa vuota** per **posizione** perché come abbiamo detto in questo caso l'utente non ha menzionato nessuna località specifica per la ricerca come "*nei dintorni*" o "*nella mia città*", e passiamo **null** per **posizione_utente** perché non abbiamo bisogno di ottenere la posizione dell'utente in questo caso.

Ma se l'utente specifica una posizione per la ricerca (es. *'nei dintorni'* o *'nella mi città'*), passiamo i parametri all'intent successivo e possiamo gestire tutto lì. l'intent successivo potrebbe essere **ottenere_posizione** or **main_ottenere_cap**, in base alla piattaforma che l'utente sta utilizzando.

```
app.intent('main', conv => {

    //specifica la piattaforma che l'utente sta utilizzando per connettersi al chatbot (es. 'google' o 'dialogflow' o 'f
    const source = conv.body.originalDetectIntentRequest.source;

    //parametri di conversazione tra chatbot e utente ('attivita', 'domanda', 'posizione' and 'categorie'). vengono da d
    let params = conv.parameters;
    let permissions = '';
    let context = '';

    if (params.posizione == '') {
        // l'utente non menziona "nei dintorni o nella mia citta o ..."
        // -non abbiamo bisogno 'permission' dell'utente per l'accesso sull'account Google
        // -chatbot risponde all'utente qui nell'intent "main"

        //RISPOSTA FINALE DI CHATBOT
        let RES = functions.risposta(params.categorie, params.domanda, params.attivita, posizione: '', posizione_utente: null);
        conv.ask(RES);
    } else {
        // l'utente menziona "nei dintorni o nella mia citta o ..."
        // -dobbiamo chiedere 'permission' per accedere all'account Google dell'utente per ottenere la sua posizione
        // -passiamo i parametri all'intent successivo

        if (source == 'google') {...} else {
            // se l'utente è connesso da un'altra piattaforma (non Google Assistant)
            // -se abbiamo gia il coordinate oppure CAP dell'utente chatbot risponde qui:
            conv.ask( responses: `Risposata di chatbot ...`);
            // -se non abbiamo coordinate oppure CAP dell'utente, passiamo i parametri all'intent "main_ottenere_cap"
            // -la conversazione va all'intent 'main_ottenere_cap'
        }
    }
});
```

main è il nome che abbiamo impostato per l'intento in dialogflow e **conv** gestisce la conversazione e i parametri, in questo intento abbiamo quattro parametri che usiamo in tutto il codice per gestire la conversazione e il chatbot risponde all'utente in base a questi parametri .

- **attivita** potrebbe essere *'mangiare'* o *'comprare'* o *'trovare'* o ….
- **domanda** potrebbe essere *'dove'* o *'quali'* o …
- **posizione** potrebbe essere *'dintorni'*, *'citta'* o *'regione'*
- **categorie** potrebbe essere *'formaggi'*, *'ricotte'*, *'taralli'* o …

conv mantiene tutti questi parametri e possiamo ottenerli da conv ogni volta che vogliamo all'interno di questo intent handler.

Se l'utente chiede al chatbot di cercare un prodotto specifico in un luogo specifico (es. *'dove posso mangiare formaggi intorno a me?*' o *'quali formaggi posso comprare nella mia città?*'), l'intent **main** passa la conversazione e tutti i parametri a questo intent, ma prima di farlo dobbiamo ottenere la posizione dell'utente dal suo account Google, per farlo dobbiamo ottenere **Permission** dall'utente per avere accesso ai suoi dati sul suo account Google. Quindi all'interno dell'intent **main** prima di passare all'intent **ottenere_posizione** inviamo un messaggio all'utente per ottenere il suo permesso di accesso ai suoi dati.

```
if (source == 'google') {
    // se l'utente è connesso da Google Assistant
    // -passiamo i parametri all'intent "ottenere_posizione"
    if (conv.user.verification === 'VERIFIED') {
        permissions = 'DEVICE_PRECISE_LOCATION'; // tipo di permission (nel questo caso : 'Posizione Preciso')
        context = 'Per sapere la tua posizione'; // prima parte del massaggio (possiamo gestire solo questa parte)
    } else {
        conv.close( responses: 'Non sei verificato');
    }
    const options = {
        context,
        permissions,
    };

    // -chiedere 'permission' all'utente per l'accesso ai suoi dati sull'account Google
    // -la conversazione va all'intent 'ottenere_posizione'
    conv.ask(new Permission(options));
```

Dopo aver ottenuto il permesso dell'utente per avere accesso ai suoi dati, possiamo ottenere la sua posizione dal suo account Google (solo quando l'utente utilizza **Google Assistant** Platform), quindi possiamo passare la conversazione all'intento **ottenere_posizione**.

ottenere_posizione (only when user is using Google Assistant Platform)

```
// gestisce la risposta del chatbot quando l'utente menziona una posizione (es. 'nella mia citta' o 'nei dintorni')
// -quando l'utente utilizza Google Assistant
app.intent('ottenere_posizione', (conv : DialogflowConversation<unknown, unknown, Contexts> , params : TParameters , confirmationGrant
    console.log(conv.body.originalDetectIntentRequest.source);
    const {location} = conv.device;
    console.log(params);

    if (confirmationGranted && location) {
        // se abbiamo accesso ai dati dell'utente dal suo account Google, il chatbot risponde qui:

        // RISPOSTA FINALE DI CHATBOT
        let RISPOSTA = functions.risposta(params.categorie, params.domanda, params.attivita, params.posizione, locat
        conv.ask(RISPOSTA);

    } else {
        // se l'utente non dà il permesso di accedere ai dati sul suo account Google a chatbot
        conv.close( responses: `non sono riuscito a capire la tua posizione.`);
    }
});
```

ottenere_posizione è il nome dell'intento in dialogflow, **params** sono i parametri della conversazione e **confirmationGranted** è un booleano e se è **true** significa che l'utente ha accettato la nostra richiesta di autorizzazione e se è **false** significa che l'utente non ci ha dato il permesso di accedere ai suoi dati sul suo account Google.

Quindi, se **confirmationGranted** è **true**, possiamo ottenere la posizione dell'utente e abbiamo anche i parametri in modo da poter creare la risposta finale del chatbot utilizzando la funzione:

risposta(params.categorie, params.domanda, params.attivita, params.posizione, location), viene da function.js :

```
function risposta(categoria, domanda, attivita, posizione, posizione_utente)
```

which we have already created in fonctions.js and imported to index.js.

location è un oggetto che include **l'indirizzo formattato** dell'utente, il **codice postale**, la **città** e le **coordinate** e che usiamo per calcolare la posizione effettiva dell'utente, la città e la regione per creare la risposta finale della base chatbot su richiesta dell'utente in **functions.js**.

e se **confirmationGranted** è **false**, significa che l'utente non ci ha dato l'autorizzazione, quindi chatbot chiude la conversazione con una risposta specifica.

Eseguire il codice

Here we run the application using express and bodyParser using the port which we have already defined it in index.js.

```
//Eseguire l'app
const expressApp = express().use(bodyParser.json());
expressApp.post( path: '/fulfillment', app);
expressApp.listen(port,  hostname: function () {
    console.log('FirstLife is runnign on ...' + port);
});
```