Detailed Documentation for GPT Web Application

Overview

gpt is a web application designed to manage interactions between a CRM system ,ChatGpt and Superchat using various APIs. This documentation provides an in-depth guide to the application, covering setup, configuration, and code functionalities.

Project Structure

- package.json
- .env
- .gitignore
- fetch.js
- generateZohoToken.js
- gptFilter.js
- index.js
- sendMessagefromAttribute.js
- superchatFunctions.js
- zohoFunctions.js

File Descriptions and Functionalities

1. package.json

This file contains metadata about the project and its dependencies.

```
{
 "name": "gpt",
 "version": "1.0.0",
 "description": "this is the first version if the easy menu web app",
 "main": "index.js",
 "type": "module",
 "scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "nodemon index.js"
 },
 "author": "Bassem",
 "license": "ISC",
 "dependencies": {
  "@ngrok/ngrok": "^1.3.0",
  "@zohocrm/nodejs-sdk-6.0": "^2.0.0",
  "axios": "^1.7.2",
  "body-parser": "^1.20.2",
  "dirname": "^0.1.0",
  "dotenv": "^16.4.5",
  "express": "^4.19.2",
  "openai": "^4.49.1"
 },
 "devDependencies": {
  "nodemon": "^3.1.2"
 }
}
```

2. .env

Contains environment variables necessary for the application to run. They are in the digtalocean or with Bassem or Robert. The env should keep stored in a save way.

Example:

```
CLIENT_ID=your_client_id
CLIENT_SECRET=your_client_secret
REDIRECT_URI=your_redirect_uri
REFRESH_TOKEN=your_refresh_token
ZOHO_OAUTH_TOKEN=your_zoho_oauth_token
SUPERCHAT_API_KEY=your_superchat_api_key
SUPERCHATCHANNEL_ID=your_superchat_channel_id
OPENAI_API_KEY=your_openai_api_key
OPENAI_ASSISTANT=your_openai_assistant
NGROK_TOKEN=your_ngrok_token
GPT_ATTRIBUTE=your_gpt_attribute
GPT_LEBEL=your_gpt_label
PORT=3000
```

3. .gitignore

Specifies files and directories to be ignored by Git.

Example:

```
node_modules
.env
```

4. fetch.js

Handles interactions with the OpenAI API and Zoho CRM. It check if the customer who sent the message has already a thread id in zoho leads, if it has it, then it take it the process it in the same thread and send the message to Superchat.

```
import axios from 'axios';
import dotenv from 'dotenv';
import { generateZohoOauthToken } from './generateZohoToken.js';
import OpenAI from "openai";
import { updateRecord, createRecord } from './zohoFunctions.js';
import { getSuperchatRecord, sendMessage } from './superchatFunctions.js';

dotenv.config();

const openai = new OpenAI();
const OPENAI_API_URL = 'https://api.openai.com/v1/';
const ZOHO_CRM_API_URL = 'https://www.zohoapis.eu/crm/v6/';
const OPENAI_API_KEY = process.env.OPENAI_API_KEY;
const OPENAI_ASSISTANT = process.env.OPENAI_ASSISTANT;
const SUPERCHAT_API_KEY = process.env.SUPERCHAT_API_KEY;

const headers = {
  'Authorization': `Bearer ${OPENAI_API_KEY}`,
  'Content-Type': 'application/json',
  'OpenAI-Beta': 'assistants=v2'
};

export async function call_in_OpenAi(mg, phone, superchat_contact_id, checker) {
  // Function implementation
}

export async function putMessageInThreadAssistant(template_id, quickReplayBody, phone)
{
  // Function implementation
}

async function getContentTemplateFromSuperchat(template_id) {
  // Function implementation
}
```

5. generateZohoToken.js

Generates and refreshes Zoho OAuth tokens.

```
import axios from 'axios';
import dotenv from 'dotenv';

dotenv.config();

const ZOHO_TOKEN_URL = 'https://accounts.zoho.eu/oauth/v2/token';
const CLIENT_ID = process.env.CLIENT_ID;
const CLIENT_SECRET = process.env.CLIENT_SECRET;
const REDIRECT_URI = process.env.REDIRECT_URI;
const REFRESH_TOKEN = process.env.REFRESH_TOKEN;

export async function generateZohoOauthToken() {
  try {
    const response = await axios.post(ZOHO_TOKEN_URL, null, {
      params: {
        client_id: CLIENT_ID,
        client_secret: CLIENT_SECRET,
        redirect_uri: REDIRECT_URI,
        refresh_token: REFRESH_TOKEN,
        grant_type: 'refresh_token'
      },
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
      }
    });

    const { access_token, expires_in } = response.data;
    process.env.ZOHO_OAUTH_TOKEN = access_token;

    return access_token;
  } catch (error) {
    console.error('Error generating Zoho OAuth token:', error.response ?
error.response.data : error.message);
    throw error;
  }
}
```

6. gptFilter.js

Filters and processes messages based on certain labels and attributes. If client hat one of the lebels , it should just store the message in the thread and do nothing.

```
import { getSuperchatRecord, getSuperchatConveration } from './superchatFunctions.js';
import { config } from 'dotenv';
import { call_in_OpenAi } from './fetch.js';
import { putMessageInThreadAssistant } from './fetch.js'

config();

const blockingLebels = [
  process.env.BLOCK_AI_LEBEL,
  process.env.VIP_KUNDE_LEBEL,
  process.env.BESTANDSKUNDE_LEBEL,
];
const gbt_attribute_id = process.env.GPT_ATTRIBUTE;
const gpt_lebel = process.env.GPT_LEBEL;

export async function runGpt(contact_id, mg , phone) {
  // Function implementation
}
```

7. index.js

Main entry point of the application. Sets up Express server and handles routes. It start with create a list of client, every client has more information like the client id, phone…. After a message arrives and it wait 20 second to process all the message those came in 20 second together. If a client has a blocker lebel then it should watch the out bound messages if a message is arrive, that's mean the automation has replayed.

```
import express from 'express';
import path from 'path';
import { fileURLToPath } from 'url';
import { dirname } from 'path';
import bodyParser from 'body-parser';
import { config } from 'dotenv';
import { call_in_OpenAi, putMessageInThreadAssistant } from './fetch.js';
import { runGpt } from './gptFilter.js';
import { runThreadAndSend } from './sendMessagefromAttribute.js';

config();
```

```
const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);

const app = express();
const PORT = process.env.PORT || 3000;
const NGROK_AUTHTOKEN = process.env.NGROK_TOKEN;

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(express.static(path.join(__dirname, 'public')));

app.get('/', (req, res) => {
   res.send('Hello World!');
});

let userInfo = {}; // To store user messages and timeouts

app.post('/webhook', (req, res) => {
   // Route implementation
});

app.post('/outboundWebhook', (req, res) => {
   // Route implementation
});

app.post('/runchatgpt', (req, res) => {
   // Route implementation
});

app.listen(PORT, () => {
  console.log('Server is running on http://localhost:' + PORT);
});
```

8. sendMessagefromAttribute.js

Handles sending messages based on specific attributes. If the gpt has been updated to 1, then it run the thread and send the message to the client.

```
import { config } from 'dotenv';
import { generateZohoOauthToken } from './generateZohoToken.js';
import OpenAI from "openai";
import axios from 'axios';
import { sendMessage } from './superchatFunctions.js';

config();

export async function runThreadAndSend(contact) {
  // Function implementation
}
```

9. superchatFunctions.js

Functions to interact with the Superchat API.

```
import { config } from 'dotenv';

config();

const SUPERCHAT_API_KEY = process.env.SUPERCHAT_API_KEY;
const SUPERCHATCHANNEL_ID = process.env.SUPERCHATCHANNEL_ID;

export async function getSuperchatRecord(contact_id) {
  // Function implementation
}

export async function sendMessage(message, contact_id) {
  // Function implementation
}



export async function getSuperchatConveration(contact_id) {
  // Function implementation
}
```

10. zohoFunctions.js

Functions to interact with the Zoho CRM.

```
import * as ZOHOCRMSDK from "@zohocrm/nodejs-sdk-6.0";
import { config } from 'dotenv';
import { getSuperchatRecord } from './superchatFunctions.js';

config();
```

```
const client_id = process.env.CLIENT_ID;
const client_secret = process.env.CLIENT_SECRET;
const redirect_url = process.env.REDIRECT_URI;
const refresh_token = process.env.REFRESH_TOKEN;

export async function initializeZohoCRM() {
  // Function implementation
}

export async function updateRecord(recordId, thread_id) {
  // Function implementation
}

export async function createRecord(phone, thread_id, superchat_contact_id) {
  // Function implementation
}
```

Setting Up the Application

Prerequisites

1. Node.js and npm installed on your system.

2. Necessary environment variables configured in a `.env` file.

Installation

1. Clone the repository.

2. Navigate to the project directory.

3. Run the following command to install dependencies:

```
npm install
```

4. if you want to run the program locally you need to set up ngrok. And add the webhook to superchat.

Running the Application

1. Start the server using the following command:

```
npm start
```

The server will start on the specified port (default is 8080).

Hosting

The program is hosted in DigtalOcean with Abrahairadabra gmail account, and connected to the digital-ecosystem github account, pushing to the main branch will update the program in the server.

API Endpoints

Webhook Endpoint

URL: `/webhook`

Method: `POST`

Description: Handles incoming messages and processes them.

Outbound Webhook Endpoint

URL: `/outboundWebhook`

Method: `POST`

Description: Handles outbound messages.

Run ChatGPT Endpoint

URL: `/runchatgpt`

Method: `POST`

Description: Runs ChatGPT for specific contacts based on attributes “GPT”.

Key Functions

Generating Zoho OAuth Token

File: `generateZohoToken.js`

Function: `generateZohoOauthToken`

Generates and refreshes the Zoho OAuth token for API authentication.

Fetch Functions

File: `fetch.js`

Functions:

- `call_in_OpenAi`

- `putMessageInThreadAssistant`

- `getContentTemplateFromSuperchat`

Handles interactions with OpenAI and Zoho CRM.

GPT Filter

File: `gptFilter.js`

Function: `runGpt`

Filters messages based on specific labels and attributes.

Superchat Functions

File: `superchatFunctions.js`

Functions:

- `getSuperchatRecord`

- `sendMessage`

- `getSuperchatConveration`

Handles interactions with the Superchat API.

Zoho Functions

File: `zohoFunctions.js`

Functions:

- `initializeZohoCRM`

- `updateRecord`

- `createRecord`

Handles interactions with Zoho CRM.

Conclusion

This documentation provides a comprehensive guide to understanding and working with the Abrahairadabra Gpt web application. Each file and function has been described in detail to assist developers in maintaining and extending the application.