APIs for OpenAI API

Get all conversations and each conversation by userId

```
/*
 * @Author: Seven Yaoching-Chi
 * @Date: 2022-11-29 14:31:01
 * @Last Modified by: Seven Yaoching-Chi
 * @Last Modified time: 2024-06-14 22:19:23
 */

const gen_sys_msg = (msg) => ({ role: "system", content: msg })
const gen_assist_msg = (msg, isFunctionCall=false) => ({ role: "assistant",
content: msg, isFunctionCall })
const gen_user_msg = (msg) => ({ role: "user", content: msg })

router.get('/conversations', authMiddleware, async (req, res) => {
  if (req.user) {
    const conversations = await Conversation.find({ userId: req.user.id });
    res.status(200).json(conversations.map(item => ({...item._doc,
messages: ''})))
  }
})

router.get('/conversation/:id', authMiddleware, async (req, res) => {
  if (req.user && req.params.id) {
    try {
      const conversation = await Conversation.findById(req.params.id);
      if (conversation) {

res.status(200).json(JSON.parse(conversation.messages).filter(message =>
message.content && message.role !== 'tool'))
      } else {
        res.status(404).json('failed')
      }
    } catch (err) {
      res.status(404).json('failed')
    }
  }
})
```

Create a new conversation for user

```
/*
 * @Author: Seven Yaoching-Chi
 * @Date: 2022-11-29 14:31:01
 * @Last Modified by: Seven Yaoching-Chi
 * @Last Modified time: 2024-06-14 22:19:23
 */

const gen_sys_msg = (msg) => ({ role: "system", content: msg })
const gen_assist_msg = (msg, isFunctionCall=false) => ({ role: "assistant",
content: msg, isFunctionCall })
const gen_user_msg = (msg) => ({ role: "user", content: msg })

const handleFunctionCall = async (openai, responseMsg, user) => {
  const funcCallMsgs = openai.functionCall(responseMsg)
  // handle the async function call for the results.
  const allMsgs = await Promise.all(funcCallMsgs.map(res => res.content))
  for (let i = 0; i < funcCallMsgs.length; i++) {
    funcCallMsgs[i].content = allMsgs[i]
  }
  console.log({user, funcCallMsgs})
  return funcCallMsgs
}

router.post('/init', authMiddleware, openaiMiddleware, async (req, res) =>
{
  const params = req.body;
  if (!isNaN(params.sysContentIndex) && !isNaN(params.personaTypeIndex) &&
req.user) {
    const {id} = req.user
    let messages = [gen_sys_msg(SYS_CONTENTS[params.sysContentIndex] +
PERSONAS[params.personaTypeIndex].content)]
    const openai = global.currentUsers[id].openAIInfo.openai
    try {
      let completion = await openai.createConversation(messages, TOOLS,
'auto')
      let responseMsg = completion.choices[0].message

      if (responseMsg.tool_calls) {
        messages.push(responseMsg)
        const funcCallMsgs = await handleFunctionCall(openai, responseMsg,
req.user)
```

```
        messages = [...messages, ...funcCallMsgs]
        completion = await openai.createConversation(messages)
        responseMsg = gen_assist_msg(completion.choices[0].message.content,
true)
      }

      messages.push(responseMsg)

      const newConversation = new Conversation({ userId: id, messages:
JSON.stringify(messages), persona: PERSONAS[params.personaTypeIndex].name,
createAt: new Date().getTime() });
      await newConversation.save()

      res.status(200).json(newConversation);
    } catch (err) {
      res.status(400).json('openAI API failed.');
    }
  } else {
    res.status(400).json('failed');
  }
});
```

User sends a new message to OpenAI

```
/*
 * @Author: Seven Yaoching-Chi
 * @Date: 2022-11-29 14:31:01
 * @Last Modified by: Seven Yaoching-Chi
 * @Last Modified time: 2024-06-14 22:19:23
 */

const gen_sys_msg = (msg) => ({ role: "system", content: msg })
const gen_assist_msg = (msg, isFunctionCall=false) => ({ role: "assistant",
content: msg, isFunctionCall })
const gen_user_msg = (msg) => ({ role: "user", content: msg })

const handleFunctionCall = async (openai, responseMsg, user) => {
  const funcCallMsgs = openai.functionCall(responseMsg)
  // handle the async function call for the results.
  const allMsgs = await Promise.all(funcCallMsgs.map(res => res.content))
  for (let i = 0; i < funcCallMsgs.length; i++) {
    funcCallMsgs[i].content = allMsgs[i]
  }
  console.log({user, funcCallMsgs})
  return funcCallMsgs
}


router.post('/msg', authMiddleware, openaiMiddleware, async (req, res) => {
  const params = req.body;
  if (req.user && params.prompt && params.conversationId) {
    try {
      const conversation = await
Conversation.findById(params.conversationId);

      let messages = JSON.parse(conversation.messages)
      messages.push(gen_user_msg(params.prompt))

      const openai = global.currentUsers[req.user.id].openAIInfo.openai
      let completion = await openai.createConversation(messages, TOOLS,
'auto')

      let responseMsg = completion.choices[0].message
      if (responseMsg.tool_calls) {
        const funcCallMsgs = await handleFunctionCall(openai, responseMsg,
```

```
req.user)
        messages.push(responseMsg)
        messages = [...messages, ...funcCallMsgs]
        completion = await openai.createConversation(messages)
        responseMsg = gen_assist_msg(completion.choices[0].message.content,
true)
      }

      messages.push(responseMsg)

      conversation.messages = JSON.stringify(messages)
      await conversation.save()

      res.status(200).json(responseMsg);
    } catch (err) {
      res.status(400).json(err.toString());
    }
  } else {
    res.status(400).json('failed');
  }

});
```

OpenAI Class

```
/*
 * @Author: Seven Yaoching-Chi
 * @Date: 2024-06-07 14:56:07
 * @Last Modified by: Seven Yaoching-Chi
 * @Last Modified time: 2024-06-18 12:44:12
 */

const {openai_model, embedding_model, model_temperature, pinecone_key,
pinecone_namespace, pinecone_index} = require('../config');
const OpenAI = require('openai');
const {RESTAURANTS} = require('../constant/constants')
const PineconeEvents = require('./pineconeEvents')

class OpenAIService {
  constructor(apiKey, model, embeddingModel, temperature) {
    this.apiKey = apiKey;
    this.openai = new OpenAI({ apiKey });
    this.model = model;
    this.embeddingModel = embeddingModel;
    this.temperature = temperature;
  }

  async createConversation(messages, tools = null, tool_choice = null) {
    try {
      return await this.openai.chat.completions.create({
        messages,
        model: this.model,
        temperature: this.temperature,
        tools,
        tool_choice
      });
    } catch (err) {
      throw new Error(err);
    }
  }

  async embeddingQuery(query) {
    try {
      const res = await this.openai.embeddings.create({
        input: JSON.stringify(query),
        model: this.embeddingModel
```

```
      });
      return res.data[0].embedding;
    } catch (err) {
      throw new Error(err);
    }
  }
}

class FunctionHandler {
  constructor() {
    this.availableFunctions = {};
  }

  registerFunction(name, func) {
    this.availableFunctions[name] = func;
  }

  async handleFunctions(toolCalls) {
    try {
      return await Promise.all(toolCalls.map(async (toolCall) => {
        const { name, arguments: args } = toolCall.function;
        const functionArgs = JSON.parse(args);
        const func = this.availableFunctions[name];
        if (func) {
          const res = await func(functionArgs);
          return {
            tool_call_id: toolCall.id,
            role: "tool",
            name,
            content: res,
          };
        } else {
          throw new Error(`Function ${name} not found`);
        }
      }));
    } catch (err) {
      console.error('Error in handleFunctions:', err);
      return [];
    }
  }
}

class RestaurantService {
```

```
  constructor(knowledgeBaseService) {
    this.knowledgeBaseService = knowledgeBaseService;
  }

  async getRestaurants({ location, type }) {
    const locationLowerCase = location.toLowerCase();
    const list = RESTAURANTS[locationLowerCase] || [];
    const restaurants = type ? list.filter(item => item.type ===
type.toUpperCase()) : list;

    if (restaurants.length === 0) {
      return await this.knowledgeBaseService.searchKnowledgeBase({
location, cuisine: type });
    } else {
      return JSON.stringify({ location, restaurants });
    }
  }
}

class ReservationService {
  makeReservation({ restaurantName, date, time, partySize }) {
    return JSON.stringify({
      restaurantName,
      date,
      time,
      partySize,
      confirmation_number: "ABC123456",
      status: "Confirmed",
    });
  }
}

class KnowledgeBaseService {
  constructor(pineconeEvents) {
    this.pineconeEvents = pineconeEvents;
  }

  async searchKnowledgeBase({ location, cuisine }) {
    console.log('Checking for knowledge base...');
    const queryMsg = { role: 'user', content: `Find the restaurants at
${location}${cuisine ? ` with the ${cuisine} cuisine type` : ''}.` };
    const xq = await this.pineconeEvents.embeddingQuery(queryMsg);
    const res = await this.pineconeEvents.queryDB({ query: xq, topK: 5 });
```

```
    const contexts = res.matches.filter(item => item.score > 0.2).map(item
=> item.metadata.text);

    const delimiter = '###';
    const limit = 3600;
    const prompt = contexts.reduce((acc, context) => {
      return acc.length + context.length < limit ? acc + context : acc;
    }, '');

    return JSON.stringify([
      { role: 'system', content: `${delimiter}${prompt}${delimiter}` },
      queryMsg
    ]);
  }
}

// Main class to bring everything together
class OpenAIEvents {
  constructor(apiKey) {
    const openAIService = new OpenAIService(apiKey, openai_model,
embedding_model, model_temperature);
    const knowledgeBaseService = new KnowledgeBaseService(new
PineconeEvents(pinecone_key, pinecone_namespace, pinecone_index));
    this.functionHandler = new FunctionHandler();
    this.functionHandler.registerFunction('getRestaurants', new
RestaurantService(knowledgeBaseService).getRestaurants.bind(new
RestaurantService(knowledgeBaseService)));
    this.functionHandler.registerFunction('makeReservation', new
ReservationService().makeReservation.bind(new ReservationService()));
    this.openAIService = openAIService;
  }

  async createConversation(messages, tools = null, tool_choice = null) {
    return this.openAIService.createConversation(messages, tools,
tool_choice);
  }

  async functionCall(responseMsg) {
    return this.functionHandler.handleFunctions(responseMsg.tool_calls);
  }
}
```

Pinecone Class

```
/*
 * @Author: Seven Yaoching-Chi
 * @Date: 2024-06-13 00:21:10
 * @Last Modified by: Seven Yaoching-Chi
 * @Last Modified time: 2024-06-14 17:40:25
 */

const { Pinecone } = require('@pinecone-database/pinecone')

class PineconeEvents {
  constructor(apiKey, dbNamespace, dbIndex) {
    this.apiKey = apiKey;
    this.pc = new Pinecone({apiKey});
    this.dbIndex = dbIndex
    this.dbNamespace = dbNamespace
    this.index = this.pc.index(dbIndex);
  }

  async queryDB ({query, topK}) {
    const res = await this.index.namespace(this.dbNamespace).query({
      topK,
      vector: query,
      includeValues: true,
      includeMetadata: true,
    });
    return res
  }
}
```