Chatgpt 리눅스 명령어
개발자 가이드

목차

> 소개

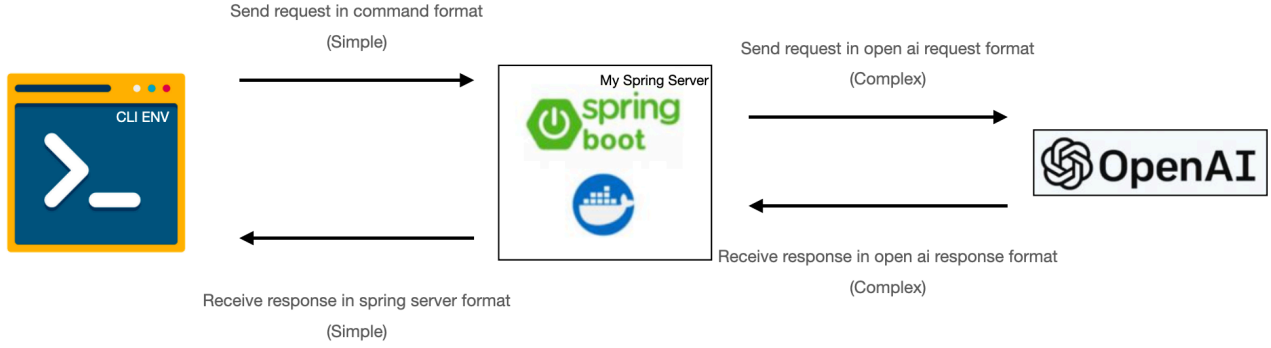
- 리눅스 환경에서 open ai 서버와 통신할 수 있는 리눅스 환경 cli 명령어를 제공합니다.
- Open ai 서버가 제공하는 다양한 api를 활용하여 사용자는 손쉽게 다양한 리눅스 명령어 기능을 사용할 수 있습니다.

> 지원하는 운영체제

	Amazon Linux	Mac OS	Ubuntu
서버	O	O	O
Amazon Web Service(AWS)	O	O	O
Google Cloud Platform(GCP)	X	O	O
Microsoft Azure	X	O	O

기본적으로 Linux 기반 운영체제인 Linux, Mac OS, Ubuntu 환경에서 사용이 가능합니다. 만약 해당 OS이외의 환경에서 해당 서비스를 사용하길 원한다면, shell 파일 안에 있는 shell script를 수정하면 됩니다.

> 프로젝트 구조



1. CLI ENV

- 사용자가 원하는 명령어를 실행하는 환경이다.
- 질문 형식의 요청을 보내면 질문에 대한 응답을 보여준다.
- UI 역할

2. Spring Server

- 사용자가 요청한 질문을 받고 이를 OpenAi Server가 원하는 형식의 요청 패킷으로 파싱한다.(OpenAiRequestEntity class)
- 서버에서 보낸 OpenAi Server 요청에 대한 응답을 사용자가 보기 편한 응답 패킷으로 파싱한다.(OpenAiResponseEntity class)

3. OpenAi Server

- 실제 Open Ai에서 운영하는 서버이다.(chatGPT 서비스를 사용할 수 있는 서버)
- 해당 서버에서는 여러 chatGPT 서비스에 대한 api를 제공한다.
- (https://platform.openai.com)에서 더 많은 api를 확인할 수 있다.

> 사용 기술

1. ObjectMapper

- JSON 형식을 사용할 때, 응답들을 직렬화하고 요청들을 역직렬화 할 때 사용하는 기술이다.
- 즉, JSON 데이터와 Java 객체 간의 변환을 수행하는 기능을 제공한다.
- Jackson 라이브러리의 클래스이다.
- JSON 데이터를 Java 객체로 변환할 때는, ObjectMapper 클래스의 readValue() 메서드를 사용한다.
- Java 객체를 JSON 데이터로 변환할 때는, ObjectMapper 클래스의 writeValue() 메서드를 사용한다.

2. ResTemplate

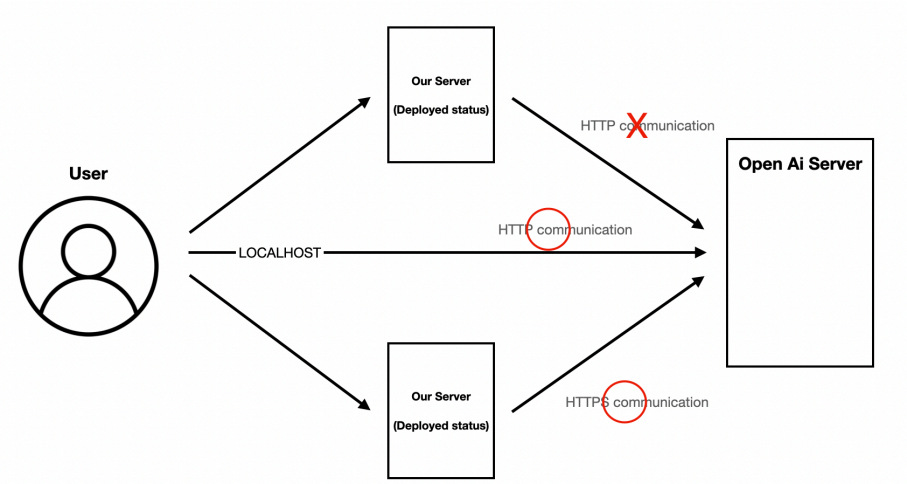
- RESTful 웹 서비스에 대한 HTTP 요청을 보내고 응답을 받아야 할때 사용하는 기술이다.
- 즉, 외부 api를 spring에서 사용할때 해당 서버에 요청을 보내고 요청에 대한 응답을 받아야 할때 사용하는 기술이다.
- RestTemplate은 간편한 방식으로 HTTP 요청을 만들고 처리할 수 있도록 다양한 메서드를 제공한다.

- 요청에 필요한 헤더, 매개변수, 본문 등을 설정할 수 있다.
- 응답을 받을 때는 응답의 상태 코드, 헤더, 본문 등을 확인할 수 있다.
- 후에 해당 기술 대신 WebClient를 사용한 기술도 추가할 예정이다.(Spring 5부터는 RestTemplate 대신 WebClient를 사용하는 것을 권장함)

3. chatGPT 질문 세션 유지 방식

- chatgpt에 질문을 하면 CLI 환경에서는 전 질문 내용이 다음 질문과 연결되지 않는 문제가 발생한다.
- 방법 1 : 질문을 연결한다. -> 질문에 대한 답이 연결이 안되서 이 또한 정상적으로 작동하지 않는다.
- 방법 2 : 질문 + 해당 질문에 대한 답을 연결한다. -> 정상적으로 작동되는 것을 확인할 수 있다.

4. 배포시 발생하는 CORS 정책 처리



- 기본적으로 OpenAI 서버는 HTTPS 통신에 개방되어 있습니다.
- 그래서 HTTP를 사용하여 OpenAI와 통신하길 원한다면, OpenAI와 통신할 수 없다.
- 그러나 OpenAI 서버가 LOCALHOST와의 통신을 허용했기 때문에 LOCALHOST를 사용하여 통신할 수 있습니다.(LOCALHOST는 기본적으로 HTTP를 사용합니다.)
- 저희는 HTTPS 통신으로 해당 문제를 해결했습니다.

> API document

3.1 질문하기

```
curl https://api.spring-chatgpt-communication.com/v1/chat/ \
  -H "Content-Type: text/plain" \
  -d '{
    what is open source?
   }'
```

위 curl 명령어를 alias를 통하여 다음과 같이 리눅스 명령어를 만들었습니다.

```
chatAsk -m "what is open source ?"
```

chatAsk 명령어를 통해 질문하기 기능을 수행할 수 있습니다.

open ai 서버로 보낼 HTTP request packet을 작성하기

```
@Bean
public HttpHeaders headers() {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    headers.setBearerAuth(token);
    return headers;
}
```

HTTP request header 부분을 작성합니다. Media type은 application/json 타입이며 Bear 에 open ai 에서 제공받은 토큰 값을 작성해 둡니다.

```java
public HttpEntity<String> chatParsed(String content) throws JsonProcessingException {
    ChatMessageDto chatMessageDto = new ChatMessageDto(Chat.ROLE.data(), content);

    String chatOpenAiBody = objectMapper.
            writeValueAsString(
                    new ChatParsedRequestDto(
                            Chat.MODEL.data(),
                            Collections.singletonList(chatMessageDto)
                    ));
    return new HttpEntity<>(chatOpenAiBody, headers);
}
```

HTTP request body 부분을 작성합니다. Body 부분에 들어갈 데이터는 모델명, messages 의 리스트 형식이 작성됩니다.

```java
public ResponseEntity<String> chatParsed(HttpEntity<String> openAiRequest) throws JsonProcessingException {
    ResponseEntity<String> response = rt.exchange(Chat.CHAT_ENDPOINT.data(), HttpMethod.POST, openAiRequest, String.class);
    OpenAiChatResponseDto openAiChatResponseDto = objectMapper.readValue(response.getBody(), OpenAiChatResponseDto.class);
    String openAiMessage = openAiChatResponseDto.getChoices().get(0).getMessage().getContent().trim();
    return getUserResponseEntity(openAiMessage);
}
```

HTTP request packet의 요청 url, HTTP method를 설정하고 restTemplate 이용하여 Open ai 서버로 요청을 보내게 됩니다. 완성된 HTTP request packet은 최종적으로 다음과 같습니다.

HTTP request packet

```
curl https://api.openai.com/v1/chat/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{
    "model": "gpt-3.5-turbo",
    "messages": [{"role": "user", "content": "what is open source?"}]
  }'
```

HTTP response packet


```
{
  "id": "chatcmpl-123",
  "object": "chat.completion",
  "created": 1677652288,
  "choices": [{
    "index": 0,
    "message": {
      "role": "assistant",
      "content": "\n\n Open source refers to a software licensing model where the source c
Freedom: Open source software provides the freedom for users to access, modify, and distri
",
    },
    "finish_reason": "stop"
  }],
  "usage": {
    "prompt_tokens": 9,
    "completion_tokens": 12,
    "total_tokens": 21
  }
}
```


해당 응답에 맞게 OpenAiChatResponseDto 클래스를 설정하였습니다

```
public ResponseEntity<String> chatParsed(HttpEntity<String> openAiRequest) throws JsonProcessingException {
    ResponseEntity<String> response = rt.exchange(Chat.CHAT_ENDPOINT.data(), HttpMethod.POST, openAiRequest, String.class);
    OpenAiChatResponseDto openAiChatResponseDto = objectMapper.readValue(response.getBody(), OpenAiChatResponseDto.class);
    String openAiMessage = openAiChatResponseDto.getChoices().get(0).getMessage().getContent().trim();
    return getUserResponseEntity(openAiMessage);
}
```

그 후 자바의 참조연산을 통해서 사용자가 필요로 하는 content 값을 제공합니다.

세션유지는 다음과 같이 진행됩니다.

```
chatRequest = chatRequest + request;
HttpEntity<String> openAiRequest = openAiRequestEntity.chatParsed(chatRequest);
ResponseEntity<String> openAiResponseEntity = this.openAiResponseEntity.chatParsed(openAiRequest);
chatRequest = chatRequest + openAiResponseEntity.getBody();
```

다음과 같이 질문 + 질문의 답변을 더하면서 다음 질문 시 이를 모두 open ai 서버에 보내며 질문 세션을 지속적으로 유지할 수 있습니다.

3.2 세션 종료

```
@GetMapping("/chat/reset")
public ResponseEntity<String> chatQuestionReset() {
    chatRequest = "";
    return ResponseEntity.ok( body: "success");
}
```

기존에 질문하기 api를 통해 질문과 질문의 답변이 누적이 되어 질문 세션이 유지가 되었던 점을 모두 초기화 시켜서 질문하기 세션을 종료하는 역할을 진행합니다.

3.3 문법 확인하기

```
curl https://api.spring-chatgpt-communication.com/v1/gc \
  -H "Content-Type: text/plain" \
  -d '{
    god morning
  }'
```

위 curl 명령어를 alias를 통하여 다음과 같이 리눅스 명령어를 만들었습니다.

```
chatAsk -gc "god morning"
```

chatAsk 명령어를 통해 문법 확인하기 기능을 수행할 수 있습니다.

open ai 서버로 보낼 HTTP request packet을 작성하기

```
@Bean
public HttpHeaders headers() {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    headers.setBearerAuth(token);
    return headers;
}
```

HTTP request header 부분을 작성합니다. Media type은 application/json 타입이며 Bear 에 open ai 에서 제공받은 토큰 값을 작성해 둡니다.

```
public HttpEntity<String> grammarCheckParsed(String prompt) throws JsonProcessingException {
    String editsOpenAiBody = objectMapper.
            writeValueAsString(
                    new CompletionsParsedRequestDto(
                            Completions.MODEL.data(),
                            prompt: Completions.GRAMMAR_CHECK.data() + prompt,
                            length: prompt.length() * 2
                    ));
    return new HttpEntity<>(editsOpenAiBody, headers);
}
```

HTTP request body 부분을 작성합니다. Body 부분에 들어갈 데이터는 모델명, prompt, prompt의 길이 정보가 작성됩니다.

```
public ResponseEntity<String> completionsParsed(HttpEntity<String> openAiRequest) throws JsonProcessingException {
    ResponseEntity<String> response = rt.exchange(Completions.ENDPOINT.data(), HttpMethod.POST, openAiRequest, String.class)
    CompletionsResponseDto completionsResponseDto = objectMapper.readValue(response.getBody(), CompletionsResponseDto.class)
    String openAiMessage = completionsResponseDto.getChoices().get(0).getText().trim();
    return getUserResponseEntity(openAiMessage);
}
```

HTTP request packet의 요청 url, HTTP method를 설정하고 restTemplate 이용하여 Open ai 서버로 요청을 보내게 됩니다. 완성된 HTTP request packet은 최종적으로 다음과 같습니다.

HTTP request packet

```
curl https://api.openai.com/v1/edits \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{
    "model": "text-davinci-edit-001",
    "input": "god morning",
    "instruction": "Fix the spelling mistakes"
  }'
```

HTTP response packet

```
{
  "object": "edit",
  "created": 1589478378,
  "choices": [
    {
      "text": "good morning",
      "index": 0,
    }
  ],
  "usage": {
    "prompt_tokens": 25,
    "completion_tokens": 32,
    "total_tokens": 57
  }
}
```

해당 응답에 맞게 CompletionsResponseDto 클래스를 설정하였습니다.

```
public ResponseEntity<String> completionsParsed(HttpEntity<String> openAiRequest) throws JsonProcessingException {
    ResponseEntity<String> response = rt.exchange(Completions.ENDPOINT.data(), HttpMethod.POST, openAiRequest, String.class);
    CompletionsResponseDto completionsResponseDto = objectMapper.readValue(response.getBody(), CompletionsResponseDto.class);
    String openAiMessage = completionsResponseDto.getChoices().get(0).getText().trim();
    return getUserResponseEntity(openAiMessage);
}
```

그 후 자바의 참조연산을 통해서 사용자가 필요로 하는 text 값을 제공합니다.

3.4 음성파일 대본 만들기

```
curl https://api.spring-chatgpt-communication.com/v1/at \
  -H "Content-Type: multipart/form-data" \
  -F file="@/path/to/file/audio.mp3" \
```

위 curl 명령어를 alias를 통하여 다음과 같이 리눅스 명령어를 만들었습니다.

```
chatAsk -at "@Desktop/음성파일.m4a"
```

chatAsk 명령어를 통해 음성파일 대본 만들기 기능을 수행할 수 있습니다.

open ai 서버로 보낼 HTTP request packet을 작성하기

```
@Bean
public HttpHeaders formHeaders() {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.MULTIPART_FORM_DATA);
    headers.setBearerAuth(token);
    return headers;
}
```

HTTP request header 부분을 작성합니다. Media type은 MULTIPART_FORM_DATA 타입이며 Bear 에 open ai 에서 제공받은 토큰 값을 작성해 둡니다.

```java
public HttpEntity<MultiValueMap<String, Object>> transcriptionParsed(MultipartFile file) {
    MultiValueMap<String, Object> requestBody = new LinkedMultiValueMap<>();
    requestBody.add("file", file.getResource());
    requestBody.add("model", Transcription.MODEL.data());

    return new HttpEntity<>(requestBody, formHeaders);
}
```

HTTP request body 부분을 작성합니다. Body 부분에 들어갈 데이터는 음성 파일, 모델명 이 작성됩니다.

```java
public ResponseEntity<String> transcriptionParsed(HttpEntity<MultiValueMap<String, Object>> openAiRequest) throws JsonProcessing
    ResponseEntity<String> response = rt.exchange(Transcription.ENDPOINT.data(), HttpMethod.POST, openAiRequest, String.class);
    TranscriptionResponseDto transcriptionResponseDto = objectMapper.readValue(response.getBody(), TranscriptionResponseDto.clas
    String openAiMessage = transcriptionResponseDto.getText();
    return getUserResponseEntity(openAiMessage);
}
```

HTTP request packet의 요청 url, HTTP method를 설정하고 restTemplate 이용하여 Open ai 서버로 요청을 보내게 됩니다. 완성된 HTTP request packet은 최종적으로 다음과 같습니다.

HTTP request packet

```
curl https://api.openai.com/v1/audio/transcriptions \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -H "Content-Type: multipart/form-data" \
  -F file="@/path/to/file/audio.mp3" \
  -F model="whisper-1"
```

HTTP response packet

```
{
  "text": "Hello, my name is Wolfgang and I come from Germany.
}
```

해당 응답에 맞게 transcriptionResponseDto 클래스를 설정하였습니다.

```
public ResponseEntity<String> transcriptionParsed(HttpEntity<MultiValueMap<String, Object>> openAiRequest) throws JsonProcessing
    ResponseEntity<String> response = rt.exchange(Transcription.ENDPOINT.data(), HttpMethod.POST, openAiRequest, String.class);
    TranscriptionResponseDto transcriptionResponseDto = objectMapper.readValue(response.getBody(), TranscriptionResponseDto.clas
    String openAiMessage = transcriptionResponseDto.getText();
    return getUserResponseEntity(openAiMessage);
}
```

그 후 자바의 참조연산을 통해서 사용자가 필요로 하는 text 값을 제공합니다.

3.5 감정분석

```
curl https://api.spring-chatgpt-communication.com/v1/mood \
  -H "Content-Type: text/plain" \
  -d '{
    i am so happy
  }'
```

위 curl 명령어를 alias를 통하여 다음과 같이 리눅스 명령어를 만들었습니다.

```
chatAsk -m "i am so happy"
```

chatAsk 명령어를 통해 문장 감정 분석 기능을 수행할 수 있습니다.

open ai 서버로 보낼 HTTP request packet을 작성하기

```
@Bean
public HttpHeaders headers() {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    headers.setBearerAuth(token);
    return headers;
}
```

HTTP request header 부분을 작성합니다. Media type은 application/json 타입이며 Bear 에 open ai 에서 제공받은 토큰 값을 작성해 둡니다.

```
public HttpEntity<String> tweetClassifierParsed(String prompt) throws JsonProcessingException {
    String tweetClassifierOpenAiBody = objectMapper.
            writeValueAsString(
                    new CompletionsParsedRequestDto(
                            Completions.MODEL.data(),
                            prompt: Completions.TWEET_CLASSIFIER.data() + prompt,
                            length: prompt.length() * 2
                    )
            );
    return new HttpEntity<>(tweetClassifierOpenAiBody, headers);
}
```

HTTP request body 부분을 작성합니다. Body 부분에 들어갈 데이터는 모델명, 문장과 prompt, prompt의 길이X2의 정보가 작성됩니다.

```
public ResponseEntity<String> completionsParsed(HttpEntity<String> openAiRequest) throws JsonProcessingException {
    ResponseEntity<String> response = rt.exchange(Completions.ENDPOINT.data(), HttpMethod.POST, openAiRequest, String.class)
    CompletionsResponseDto completionsResponseDto = objectMapper.readValue(response.getBody(), CompletionsResponseDto.class)
    String openAiMessage = completionsResponseDto.getChoices().get(0).getText().trim();
    return getUserResponseEntity(openAiMessage);
}
```

HTTP request packet의 요청 url, HTTP method를 설정하고 restTemplate 이용하여 Open ai 서버로 요청을 보내게 됩니다. 완성된 HTTP request packet은 최종적으로 다음과 같습니다.

HTTP request packet

```
curl https://api.openai.com/v1/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{
  "model": "text-davinci-003",
  "prompt": "Decide whether a Tweet's sentiment is positive, neutral, or negative.\n\n
  Tweet: \"i am so happy\"\nSentiment:",
  "temperature": 0,
  "max_tokens": 60,
  "top_p": 1.0,
  "frequency_penalty": 0.5,
  "presence_penalty": 0.0
}'
```

HTTP response packet

```
{
    "text" : "positive"
}
```

해당 응답에 맞게 CompletionsResponseDto 클래스를 설정하였습니다.

```
public ResponseEntity<String> completionsParsed(HttpEntity<String> openAiRequest) throws JsonProcessingException {
    ResponseEntity<String> response = rt.exchange(Completions.ENDPOINT.data(), HttpMethod.POST, openAiRequest, String.class);
    CompletionsResponseDto completionsResponseDto = objectMapper.readValue(response.getBody(), CompletionsResponseDto.class);
    String openAiMessage = completionsResponseDto.getChoices().get(0).getText().trim();
    return getUserResponseEntity(openAiMessage);
}
```

그 후 자바의 참조연산을 통해서 사용자가 필요로 하는 text 값을 제공합니다.

3.6 문장 번역

```
curl https://api.spring-chatgpt-communication.com/v1/trans \
  -H "Content-Type: text/plain" \
  -d '{
    hi, this is seo ji hyeon
  }'
```

위 curl 명령어를 alias를 통하여 다음과 같이 리눅스 명령어를 만들었습니다.

```
chatAsk -t "hi, this is seo ji hyeon"
```

chatAsk 명령어를 통해 문장 번역 기능을 수행할 수 있습니다.

open ai 서버로 보낼 HTTP request packet을 작성하기

```
@Bean
public HttpHeaders headers() {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    headers.setBearerAuth(token);
    return headers;
}
```

HTTP request header 부분을 작성합니다. Mine type은 application/json 타입이며 Bear 에 open ai 에서 제공받은 토큰 값을 작성해 둡니다.

```
public HttpEntity<String> translateParsed(String prompt) throws JsonProcessingException {
    String translateOpenAiBody = objectMapper.
            writeValueAsString(
                    new CompletionsParsedRequestDto(
                            Completions.MODEL.data(),
                            prompt: Completions.TRANSLATE.data() + prompt,
                            length: prompt.length() * 2
                    )
            );
    return new HttpEntity<>(translateOpenAiBody, headers);
}
```

HTTP request body 부분을 작성합니다. Body 부분에 들어갈 데이터는 모델명, 문장과 prompt, prompt의 길이X2의 정보가 작성됩니다.

```
public ResponseEntity<String> completionsParsed(HttpEntity<String> openAiRequest) throws JsonProcessingException {
    ResponseEntity<String> response = rt.exchange(Completions.ENDPOINT.data(), HttpMethod.POST, openAiRequest, String.class);
    CompletionsResponseDto completionsResponseDto = objectMapper.readValue(response.getBody(), CompletionsResponseDto.class);
    String openAiMessage = completionsResponseDto.getChoices().get(0).getText().trim();
    return getUserResponseEntity(openAiMessage);
}
```

HTTP request packet의 요청 url, HTTP method를 설정하고 restTemplate 이용하여 Open ai 서버로 요청을 보내게 됩니다. 완성된 HTTP request packet은 최종적으로 다음과 같습니다.

HTTP request packet

```
curl https://api.openai.com/v1/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{
  "model": "text-davinci-003",
  "prompt": "Translate this into korean\n\nhi, this is set ji hyeon\n\n.",
  "temperature": 0.3,
  "max_tokens": 100,
  "top_p": 1.0,
  "frequency_penalty": 0.0,
  "presence_penalty": 0.0
}'
```

HTTP response packet

```
{
  "text": "안녕하세요 저는 서지현 입니다."
}
```

해당 응답에 맞게 CompletionsResponseDto 클래스를 설정하였습니다.

```
public ResponseEntity<String> completionsParsed(HttpEntity<String> openAiRequest) throws JsonProcessingException {
    ResponseEntity<String> response = rt.exchange(Completions.ENDPOINT.data(), HttpMethod.POST, openAiRequest, String.class);
    CompletionsResponseDto completionsResponseDto = objectMapper.readValue(response.getBody(), CompletionsResponseDto.class);
    String openAiMessage = completionsResponseDto.getChoices().get(0).getText().trim();
    return getUserResponseEntity(openAiMessage);
}
```

그 후 자바의 참조연산을 통해서 사용자가 필요로 하는 text 값을 제공합니다.

3.7 문장 요약

```
curl https://api.spring-chatgpt-communication.com/v1/summarize \
  -H "Content-Type: text/plain" \
  -d '{
        i love movie, i love soccer
  }'
```

위 curl 명령어를 alias를 통하여 다음과 같이 리눅스 명령어를 만들었습니다.

```
chatAsk -s "i love movie, i love soccer"
```

chatAsk 명령어를 통해 문장 요약 기능을 수행할 수 있습니다.

open ai 서버로 보낼 HTTP request packet을 작성하기

```
@Bean
public HttpHeaders headers() {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    headers.setBearerAuth(token);
    return headers;
}
```

HTTP request header 부분을 작성합니다. Media type은 application/json 타입이며 Bear 에 open ai 에서 제공받은 토큰 값을 작성해 둡니다.

```
public HttpEntity<String> summarizeParsed(String prompt) throws JsonProcessingException {
    String summarizeOpenAiBody = objectMapper.
            writeValueAsString(
                    new CompletionsParsedRequestDto(
                            Completions.MODEL.data(),
                            prompt: prompt + Completions.SUMMARIZE.data(),
                            length: prompt.length() * 2
                    )
            );
    return new HttpEntity<>(summarizeOpenAiBody, headers);
}
```

HTTP request body 부분을 작성합니다. Body 부분에 들어갈 데이터는 모델명, 문장과 prompt, prompt의 길이X2의 정보가 작성됩니다.

```
public ResponseEntity<String> completionsParsed(HttpEntity<String> openAiRequest) throws JsonProcessingException {
    ResponseEntity<String> response = rt.exchange(Completions.ENDPOINT.data(), HttpMethod.POST, openAiRequest, String.class);
    CompletionsResponseDto completionsResponseDto = objectMapper.readValue(response.getBody(), CompletionsResponseDto.class);
    String openAiMessage = completionsResponseDto.getChoices().get(0).getText().trim();
    return getUserResponseEntity(openAiMessage);
}
```

HTTP request packet의 요청 url, HTTP method를 설정하고 restTemplate 이용하여 Open ai 서버로 요청을 보내게 됩니다. 완성된 HTTP request packet은 최종적으로 다음과 같습니다.

HTTP request packet

```
curl https://api.openai.com/v1/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{
  "model": "text-davinci-003",
  "prompt": "Summarize this for a second-grade student:\n\n i love movie and soccer",
  "temperature": 1,
  "max_tokens": 64,
  "top_p": 1.0,
  "frequency_penalty": 0.0,
  "presence_penalty": 0.0
}'
```

HTTP response packet

```
{
    "text" : "i love movie and soccer"
}
```

해당 응답에 맞게 CompletionsResponseDto 클래스를 설정하였습니다.

```
public ResponseEntity<String> completionsParsed(HttpEntity<String> openAiRequest) throws JsonProcessingException {
    ResponseEntity<String> response = rt.exchange(Completions.ENDPOINT.data(), HttpMethod.POST, openAiRequest, String.class);
    CompletionsResponseDto completionsResponseDto = objectMapper.readValue(response.getBody(), CompletionsResponseDto.class);
    String openAiMessage = completionsResponseDto.getChoices().get(0).getText().trim();
    return getUserResponseEntity(openAiMessage);
}
```

그 후 자바의 참조연산을 통해서 사용자가 필요로 하는 text 값을 제공합니다.

3.8 상위 키워드 추출

질문하기 api에서는 질문한 문장을 토대로 키워드를 데이터베이스에 저장합니다. 상위 키워드 추출 api는 저장된 키워드중 사용자가 가장 많이 질문한 키워드 상위 5개를 제공해 주는 api입니다.

```
public List<String> popularKeywords() {
    return keywordRepository.findTop5ByCountDesc(PageRequest.of( page: 0,  size: 5));
}
```

다음과 같이 SpringDataJpa의 특징 페이지네이션을 통해 상위 키워드 5개를 호출 할 수 있습니다.

> 참조 문헌

- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html

- https://platform.openai.com/examples

감사합니다.