**Generating an API key in Gemini**

1. Go to this url - <https://aistudio.google.com/apikey>
2. On top right click on create API key
3. Click on Cceate api key for new project
4. Copy that key and keep that key safe

Api key - AIzaSyABLb1Ua7MpaXD2q68EBBL7O7NvGZNVVO0

**Postman**

1. Open postman api and create a new collection
2. Select that collection click on import and paste the cURL which is present in - <https://aistudio.google.com/apikey>
3. A post method will be generated and under params in place of key give any key name (e.g key) and in place of value give the api key
4. Under body we can see the content and click on send then we can see the response

**Integrating with vite+react**

1. Install node.js
2. In vscode in terminal navigate to the project folder
3. Check for node -v and npm -v
4. After we get version enter npm create vite@latest
5. Click enter and give project name(email-writer-react)
6. Select framework as react and variant as Javascript
7. Enter the following commands
8. cd email-writer-react
9. npm install
10. npm run dev

**Project details**

1. The gemini api accepts request in this format only
   1. {
   2. "contents": [
   3. {
   4. "parts": [
   5. {
   6. "text": "Explain how AI works in a few words"
   7. }
   8. ]
   9. }
   10. ]
   11. }
2. String response = webClient.post()

.uri(geminiApiUrl + geminiApiKey)

.header("Content-Type", "application/json")

.retrieve()

.bodyToMono(String.class)

.block();

🧠 In Simple Terms:

This line is using Spring WebClient to make a synchronous HTTP POST request to the Gemini API and get the response as a string.

🔍 Detailed Breakdown:

webClient.post() ->Initiates an HTTP POST request.

.uri(geminiApiUrl + geminiApiKey) -> Sets the full URL for the request by concatenating the geminiApiUrl and geminiApiKey. ❗⚠️ This is not ideal, as the API key should be passed in a header or query param, not appended directly to the URL like this.

.header("Content-Type", "application/json") ->Sets the Content-Type header to application/json, telling the server you're sending JSON data.

.retrieve() -> Sends the request and prepares to extract the response.

.bodyToMono(String.class) ->Converts the response body into a Mono<String> (i.e., a reactive wrapper for an async String response).

.block() ->Makes the call synchronous, i.e., waits for the result and returns the actual String response.

1. ObjectMapper mapper = new ObjectMapper();

JsonNode rootNode = mapper.readTree(response);

return rootNode.path("candidates")

.get(0)

.path("content")

.path("parts")

.get(0)

.path("text")

.asText();

🔍 What It Does Step-by-Step:

ObjectMapper mapper = new ObjectMapper();

Creates a new Jackson ObjectMapper to parse JSON.

JsonNode rootNode = mapper.readTree(response);

Parses the JSON response string into a tree of JsonNode objects.

Navigate the JSON structure:

{

"candidates": [

{

"content": {

"parts": [

{

"text": "YOUR\_DESIRED\_OUTPUT"

}

]

}

}

]

}

rootNode.path("candidates") → gets the candidates array.

.get(0) → gets the first candidate object.

.path("content") → gets the content object inside the first candidate.

.path("parts") → gets the parts array inside content.

.get(0) → gets the first part.

.path("text") → gets the "text" field.

.asText() → converts that field to a plain String.

✅ So in Simple Terms:

This code extracts:

"candidates[0].content.parts[0].text"

from the JSON response returned by Gemini API or another similar LLM.

🧠 Example Output:

If this was the JSON:

{

"candidates": [

{

"content": {

"parts": [

{

"text": "Hello, this is Gemini's response."

}

]

}

}

]

}

Then the method returns:

"Hello, this is Gemini's response."

1. In **Java**, @AllArgsConstructor is a **Lombok annotation** that automatically generates a **constructor with one parameter for each field** in the class.

**✅ Example:**

import lombok.AllArgsConstructor;

@AllArgsConstructor

public class User {

private String name;

private int age;

private String email;

}

**🔨 Lombok generates this constructor:**

public User(String name, int age, String email) {

this.name = name;

this.age = age;

this.email = email;

}

**🔍 Key Points**

| **Feature** | **Description** |
| --- | --- |
| 📦 From | lombok.AllArgsConstructor |
| 🧱 What it does | Creates a constructor with **all fields as arguments** |
| 🧰 Useful when | You want to initialize all fields at once (e.g. in tests, DTOs, models) |
| 🚫 Doesn’t include | Getters/setters – you still need @Getter, @Setter, etc. separately |
| 🔄 Works with | @Data, @Builder, @NoArgsConstructor, etc. |

**💡 Tip:**

If your class has final fields, @AllArgsConstructor is especially useful because final fields must be initialized via constructor.

**🧪 Example with Lombok + Spring Boot:**

@Data

@AllArgsConstructor

@NoArgsConstructor

public class UserDto {

private Long id;

private String name;

private String email;

}

Now you can do:

java

Copy code

UserDto user = new UserDto(1L, "Sneha", "sneha@example.com");