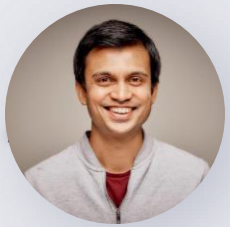
PyData Southampton

OpenAI's Function Calling: What is it and how can we take advantage of it?

Bio.tsx

```
return(
        <div>
                <img src="
                <Name>Anoop Tatti</Name>
                <Work>MVP, Developer, Advania UK</Work>
                <Profile link="aka.ms/anoopt" />
                <Blog link="https://anoopt.medium.com" />
                <Twitter username="anooptells" />
                <GitHub username="anoopt" />
        </div>
);
```



1

Introduction

OpenAI, setup, using OpenAI API

2

Function Calling

What, why and how

3

Code

Typescript code (can be changed)

4

Conclusion

Summary and resources

Introduction

AI everywhere.
- Feed

Helpful to users in many ways.
- Copilot
- ChatGPT

APIs are provided that we can use in our applications.

OpenAI and Azure OpenAI

OpenAI is an independent AI research laboratory.

Conducts fundamental research in AI and develops advanced AI models.

Focused on advancing the field of AI as a whole.

Provides API.

Azure OpenAI, on the other hand, is a partnership between OpenAI and Azure.

Provides developers with access to OpenAI's models through Azure.

Azure OpenAI is primarily focused on providing AI solutions to businesses through the Azure platform.

Provides API.

* **Before doing anything with production data, read data policies and discuss with your company's legal department.**

OpenAI API

A tool that provides access to advanced AI models through simple interface.

Helps easily integrate AI capabilities into applications.

Uses deep learning algorithm to provide high quality responses to user queries.

Can be accessed through a variety of programming languages.

Setting up OpenAI API

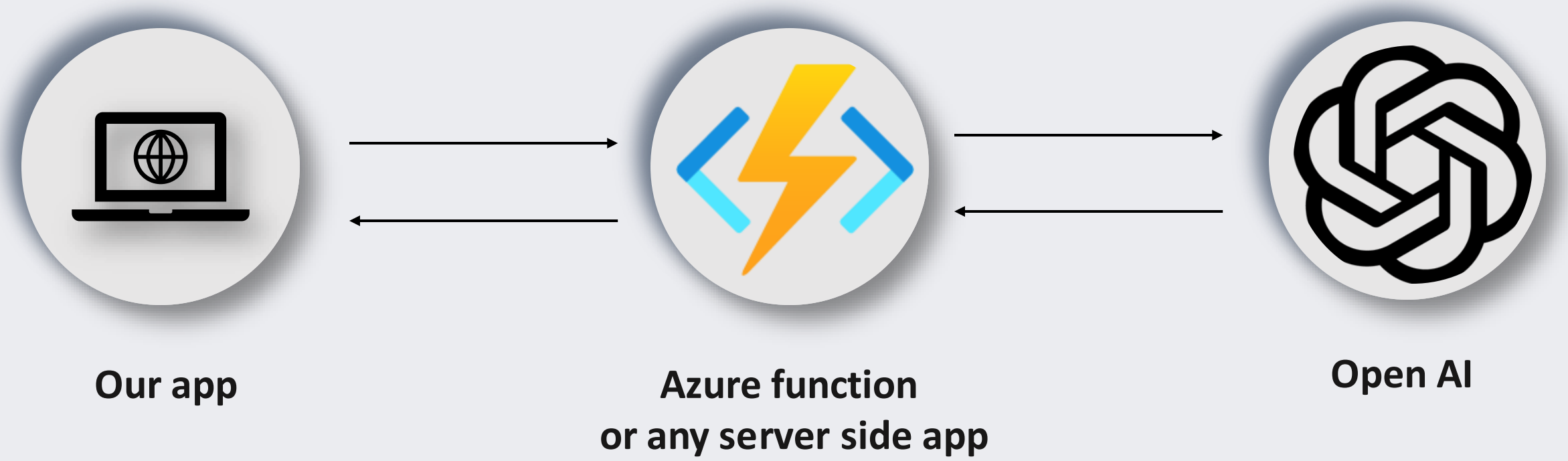
Create an OpenAI account

Obtain an API key

Playground

Demo

Integrating OpenAI API with our apps



Integrating OpenAI API with our apps

Our app

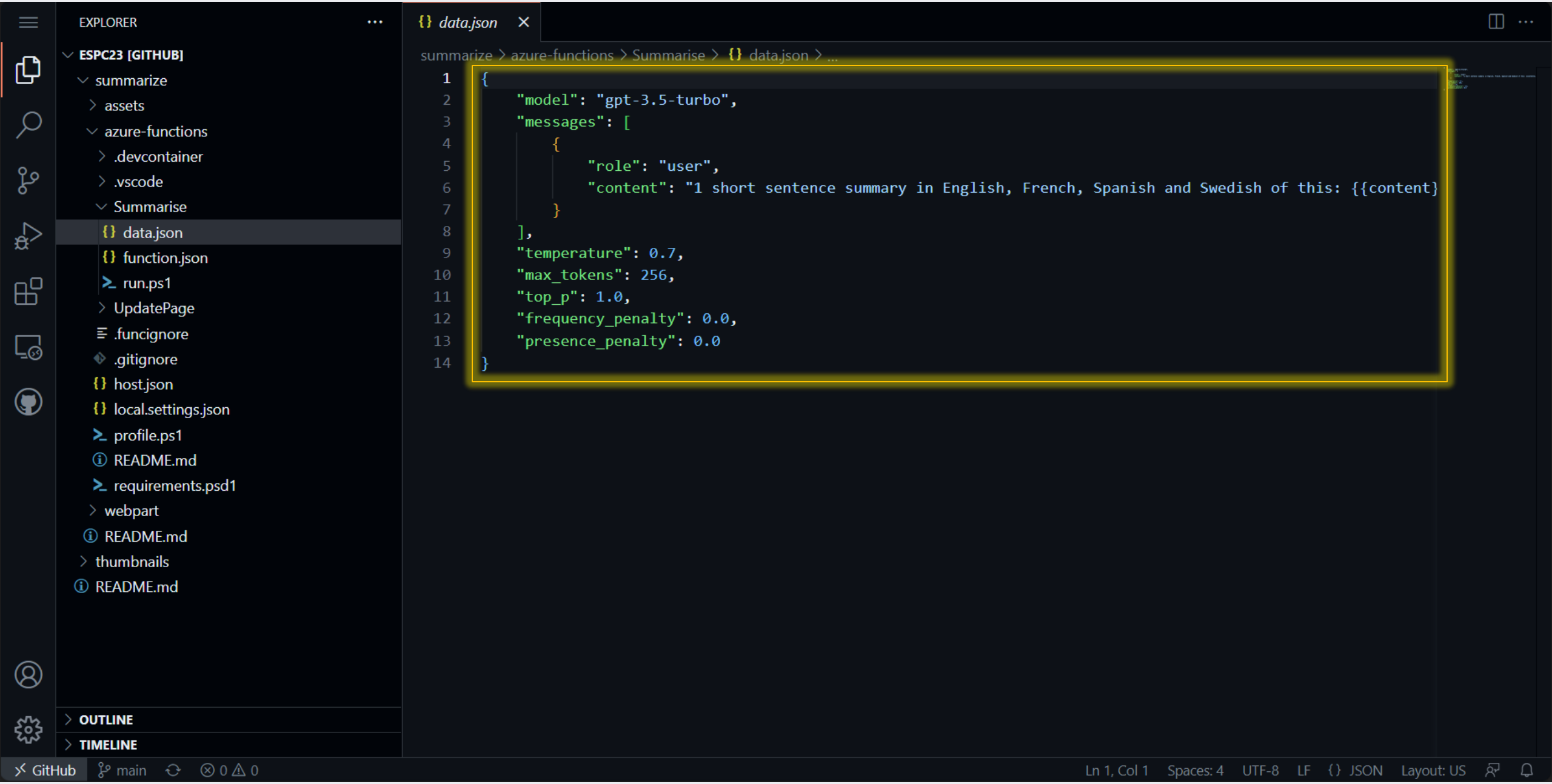
- Gets data
- Passes data Azure function
- Displays formatted data

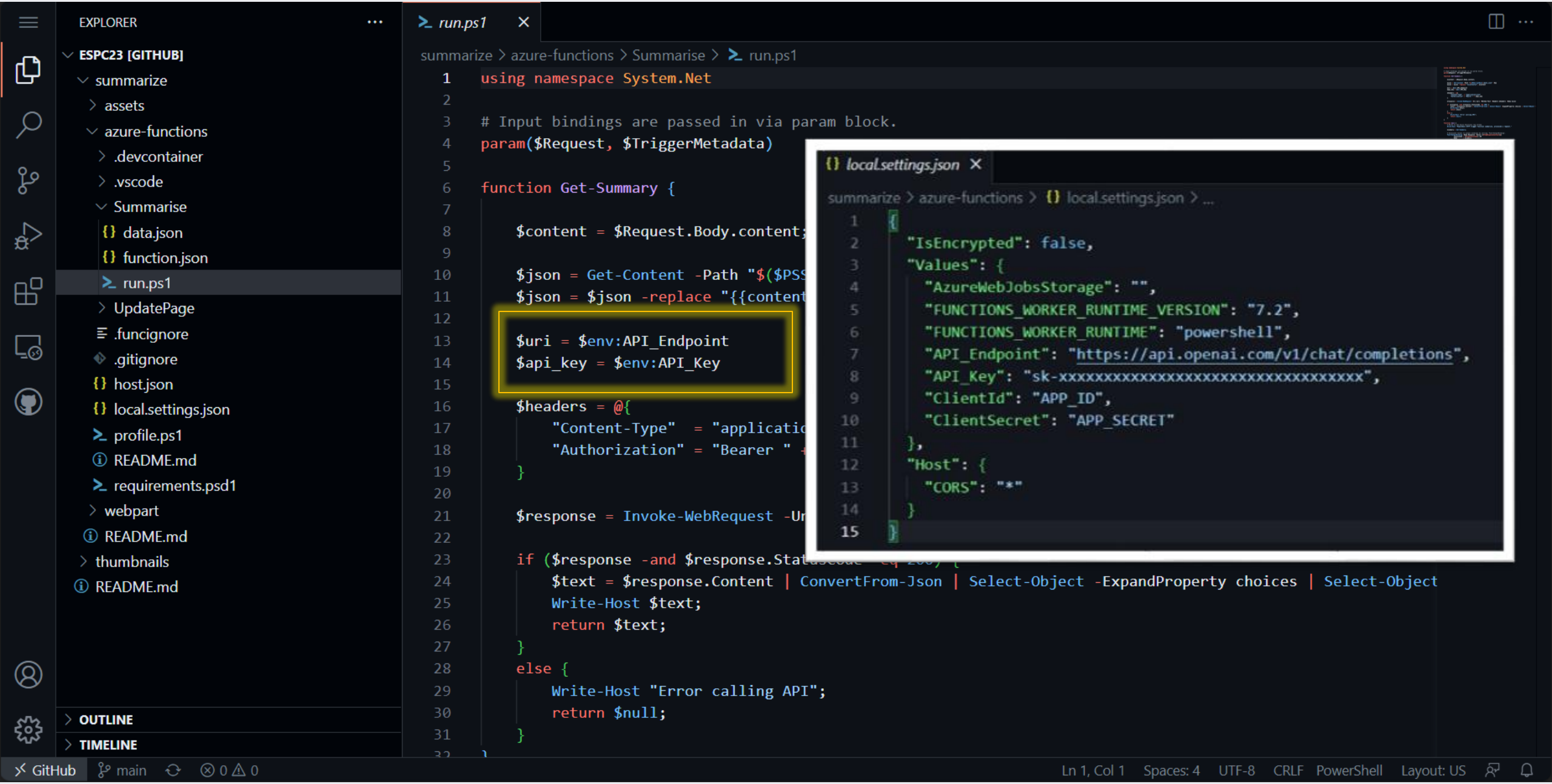
Azure function or any server side app

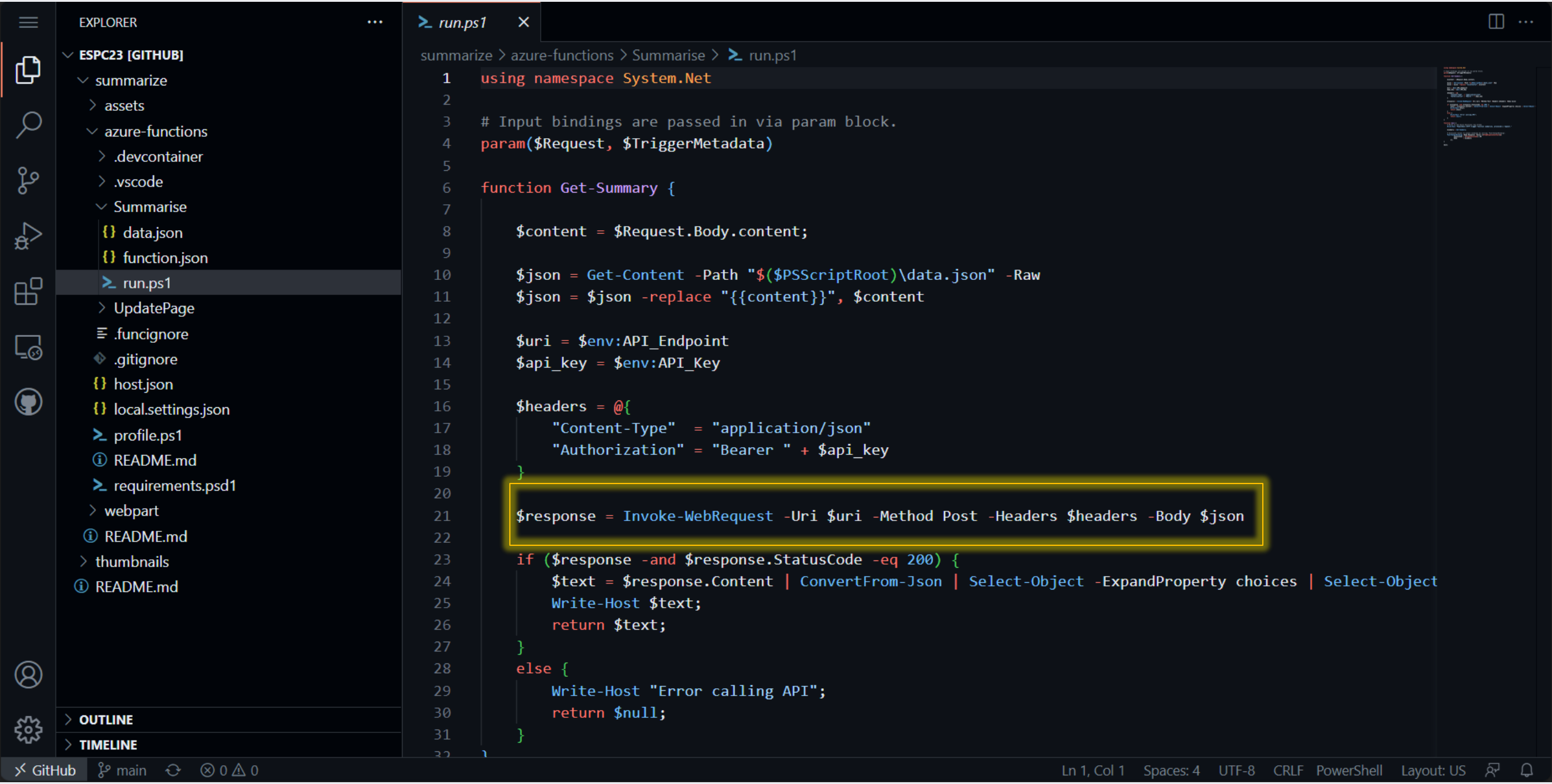
- Constructs a prompt with the data
- Calls OpenAI API
- Gets formatted data

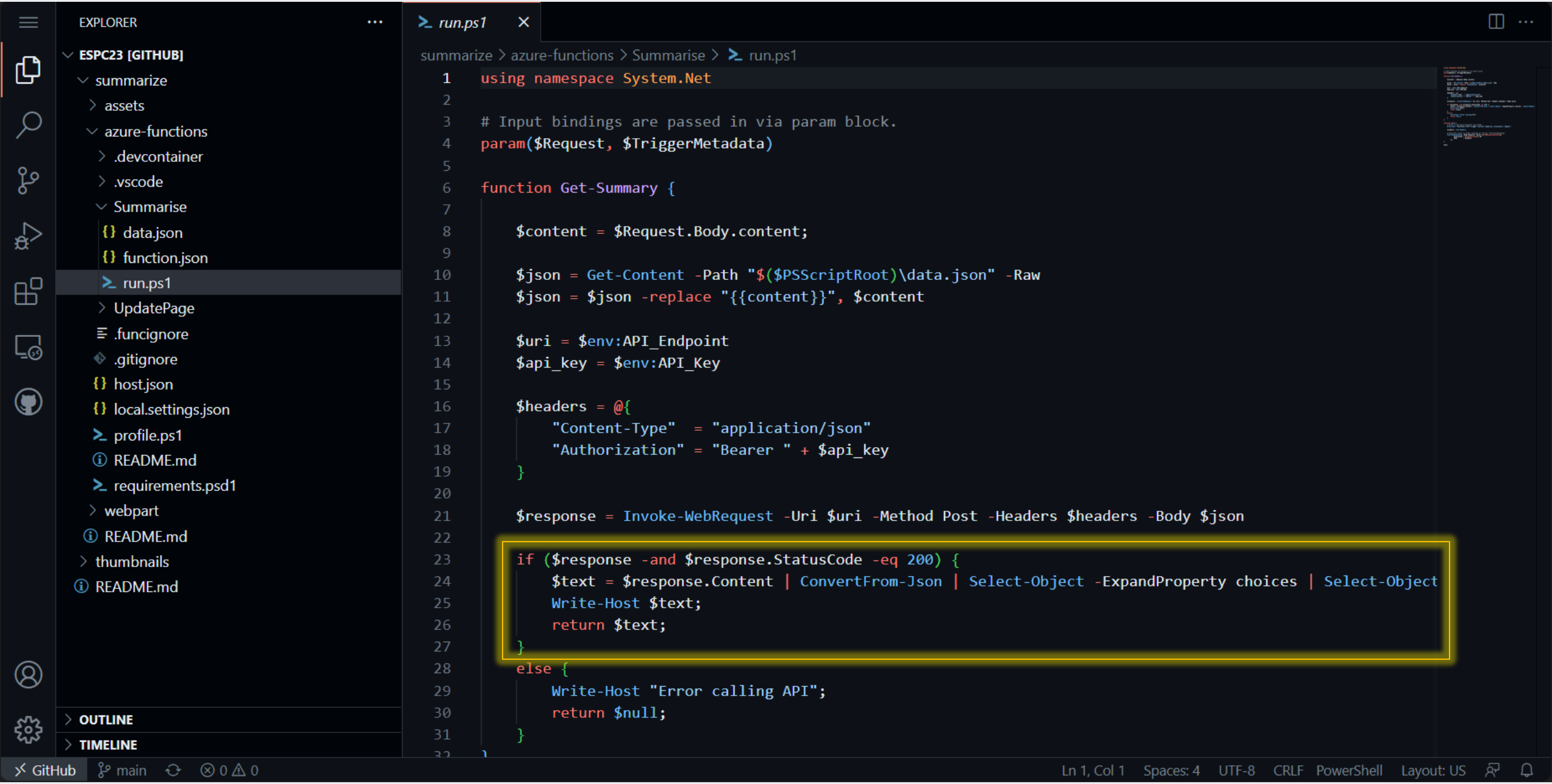
Open AI

- Uses the prompt and provides high quality response


EXPLORER
ESPC23 [GITHUB]
summarize
assets
azure-functions
.devcontainer
.vscode
Summarise
data.json
function.json
run.ps1
UpdatePage
.funcignore
.gitignore
host.json
local.settings.json
profile.ps1
README.md
requirements.psd1
webpart
README.md
thumbnails
README.md
OUTLINE
TIMELINE
data.json
summarize > azure-functions > Summarise > data.json > ...
{
"model": "gpt-3.5-turbo",
"messages": [
{
"role": "user",
"content": "1 short sentence summary in English, French, Spanish and Swedish of this: {{content}
}
],
"temperature": 0.7,
"max_tokens": 256,
"top_p": 1.0,
"frequency_penalty": 0.0,
"presence_penalty": 0.0
}
GitHub
main
0 0
Ln 1, Col 1
Spaces: 4
UTF-8
LF
{} JSON
Layout: US


EXPLORER
ESPC23 [GITHUB]
summarize
assets
azure-functions
.devcontainer
.vscode
Summarise
data.json
function.json
run.ps1
UpdatePage
.funcignore
.gitignore
host.json
local.settings.json
profile.ps1
README.md
requirements.psd1
webpart
README.md
thumbnails
README.md
OUTLINE
TIMELINE
run.ps1
summarize > azure-functions > Summarise > run.ps1
using namespace System.Net
# Input bindings are passed in via param block.
param($Request, $TriggerMetadata)
function Get-Summary {
$content = $Request.Body.content;
$json = Get-Content -Path "$($PSS
$json = $json -replace "{{content
$uri = $env:API_Endpoint
$api_key = $env:API_Key
$headers = @{
"Content-Type" = "applicatio
"Authorization" = "Bearer " +
}
$response = Invoke-WebRequest -Ur
if ($response -and $response.Statuscode -eq 200) {
$text = $response.Content | ConvertFrom-Json | Select-Object -ExpandProperty choices | Select-Object
Write-Host $text;
return $text;
}
else {
Write-Host "Error calling API";
return $null;
}
}
local.settings.json
summarize > azure-functions > local.settings.json > ...
{
"IsEncrypted": false,
"Values": {
"AzureWebJobsStorage": "",
"FUNCTIONS_WORKER_RUNTIME_VERSION": "7.2",
"FUNCTIONS_WORKER_RUNTIME": "powershell",
"API_Endpoint": "https://api.openai.com/v1/chat/completions",
"API_Key": "sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
"ClientId": "APP_ID",
"ClientSecret": "APP_SECRET"
},
"Host": {
"CORS": "*"
}
}
GitHub
main
0 0
Ln 1, Col 1
Spaces: 4
UTF-8
CRLF
PowerShell
Layout: US


EXPLORER
ESPC23 [GITHUB]
summarize
assets
azure-functions
.devcontainer
.vscode
Summarise
data.json
function.json
run.ps1
UpdatePage
.funcignore
.gitignore
host.json
local.settings.json
profile.ps1
README.md
requirements.psd1
webpart
README.md
thumbnails
README.md
OUTLINE
TIMELINE
run.ps1
summarize > azure-functions > Summarise > run.ps1
using namespace System.Net
# Input bindings are passed in via param block.
param($Request, $TriggerMetadata)
function Get-Summary {
$content = $Request.Body.content;
$json = Get-Content -Path "$($PSScriptRoot)\data.json" -Raw
$json = $json -replace "{{content}}", $content
$uri = $env:API_Endpoint
$api_key = $env:API_Key
$headers = @{
"Content-Type" = "application/json"
"Authorization" = "Bearer " + $api_key
}
$response = Invoke-WebRequest -Uri $uri -Method Post -Headers $headers -Body $json
if ($response -and $response.StatusCode -eq 200) {
$text = $response.Content | ConvertFrom-Json | Select-Object -ExpandProperty choices | Select-Object
Write-Host $text;
return $text;
}
else {
Write-Host "Error calling API";
return $null;
}
}
GitHub
main
0 0
Ln 1, Col 1
Spaces: 4
UTF-8
CRLF
PowerShell
Layout: US


EXPLORER
ESPC23 [GITHUB]
summarize
assets
azure-functions
.devcontainer
.vscode
Summarise
data.json
function.json
run.ps1
UpdatePage
.funcignore
.gitignore
host.json
local.settings.json
profile.ps1
README.md
requirements.psd1
webpart
README.md
thumbnails
README.md
OUTLINE
TIMELINE
run.ps1
summarize > azure-functions > Summarise > run.ps1
using namespace System.Net
# Input bindings are passed in via param block.
param($Request, $TriggerMetadata)
function Get-Summary {
$content = $Request.Body.content;
$json = Get-Content -Path "$($PSScriptRoot)\data.json" -Raw
$json = $json -replace "{{content}}", $content
$uri = $env:API_Endpoint
$api_key = $env:API_Key
$headers = @{
"Content-Type" = "application/json"
"Authorization" = "Bearer " + $api_key
}
$response = Invoke-WebRequest -Uri $uri -Method Post -Headers $headers -Body $json
if ($response -and $response.StatusCode -eq 200) {
$text = $response.Content | ConvertFrom-Json | Select-Object -ExpandProperty choices | Select-Object
Write-Host $text;
return $text;
}
else {
Write-Host "Error calling API";
return $null;
}
GitHub
main
0 0
Ln 1, Col 1
Spaces: 4
UTF-8
CRLF
PowerShell
Layout: US

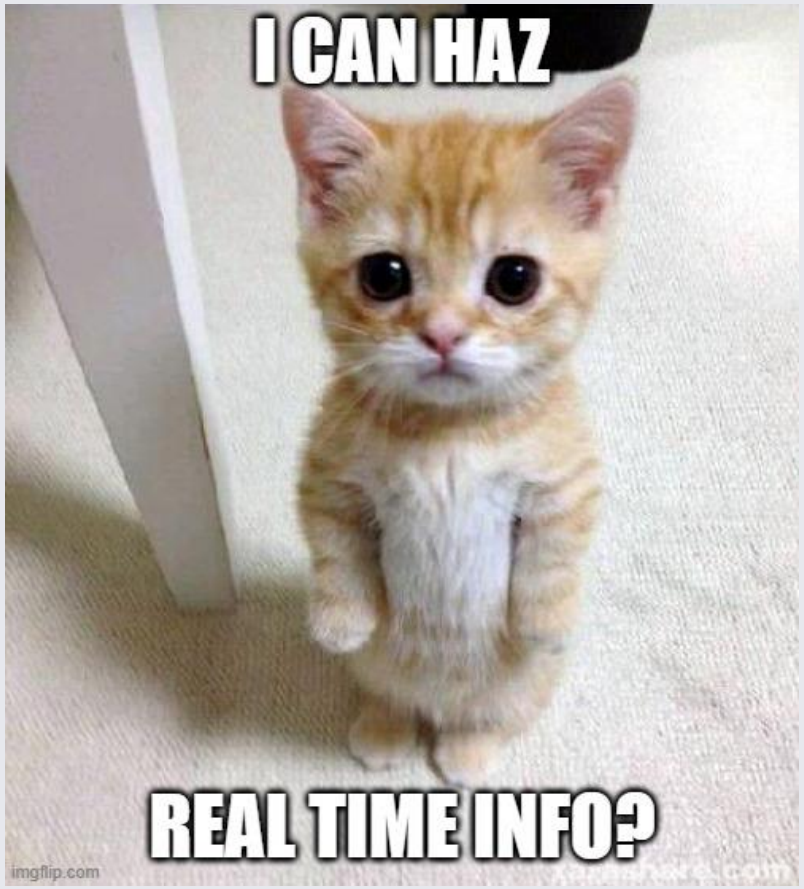
Use cases

Summarise SharePoint page content in multiple languages.

Chat bot.

Show possible thumbnails for a page.

New use cases



New use cases

Get the current weather in Southampton.

Get my next meeting details.

Status of a train.

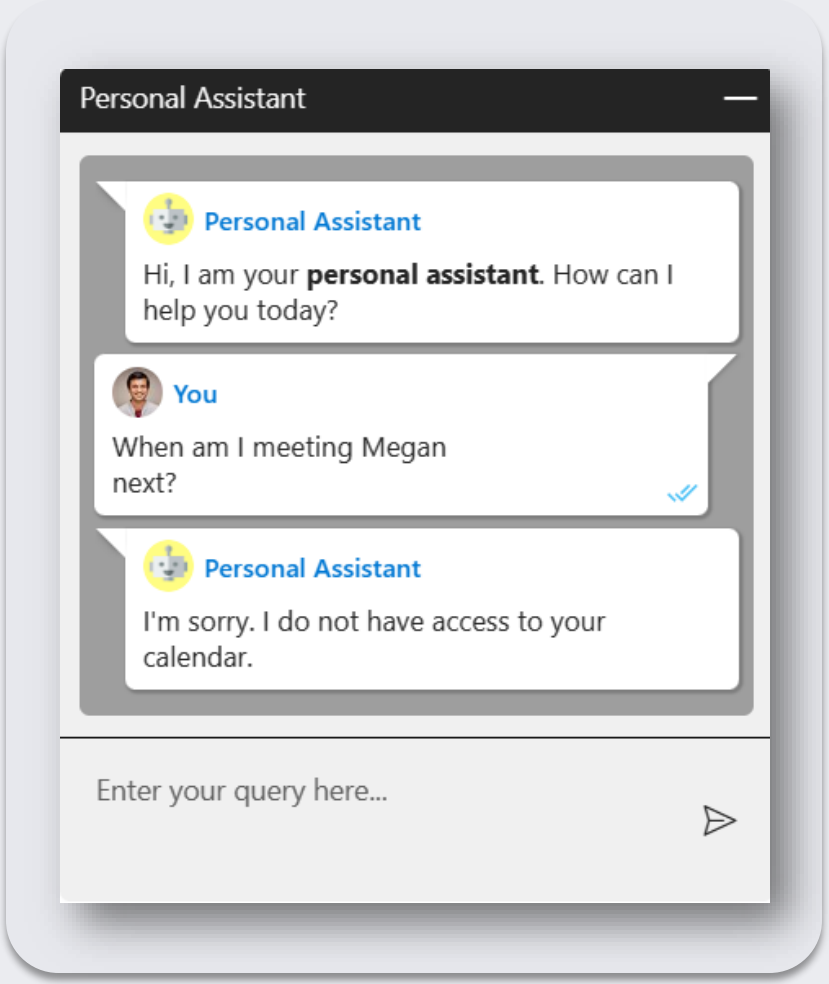
Demo

OpenAI Function Calling

- Extend OpenAI models (3.5 turbo and 4)
- Understands user intent
- E.g.: OpenAI models can’t get current weather
- Inform OpenAI models our code has functions
- OpenAI tells us which function to call (JSON containing arguments)

Personal assistant

OpenAI Function Calling



Custom app uses OpenAI API

When am I meeting Megan next?

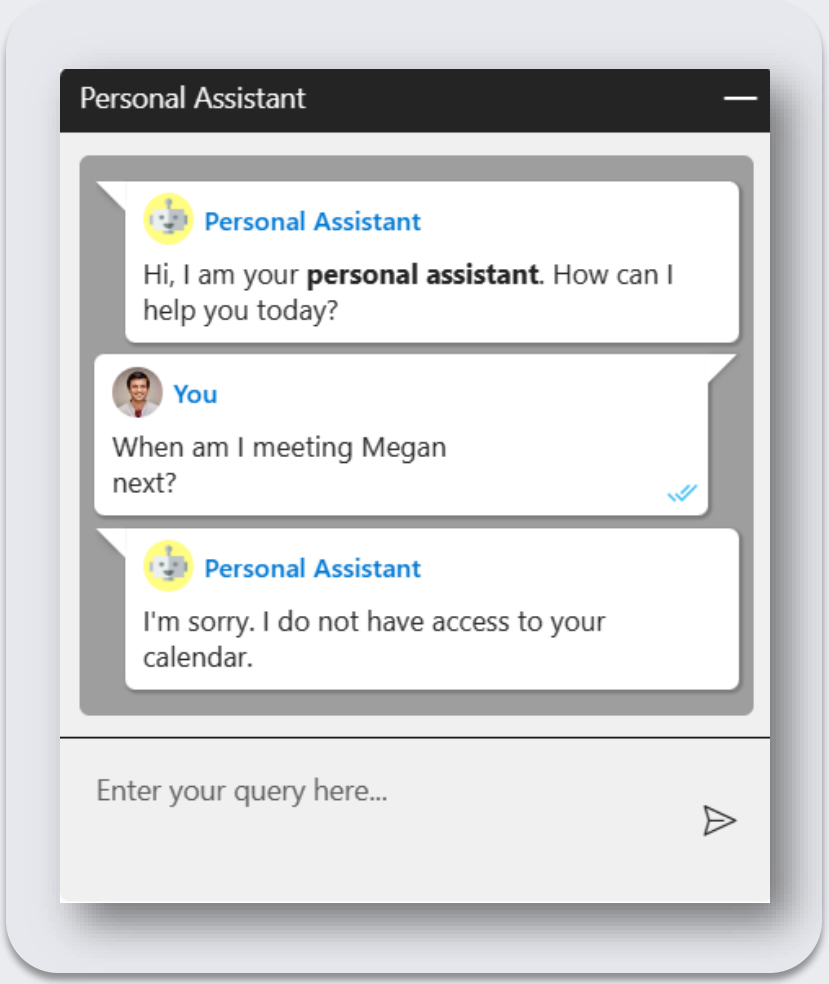
Text definition of

getMyDetails,
getMyEvents,
getMyTasks

Along with arguments and their description

OpenAI

OpenAI Function Calling



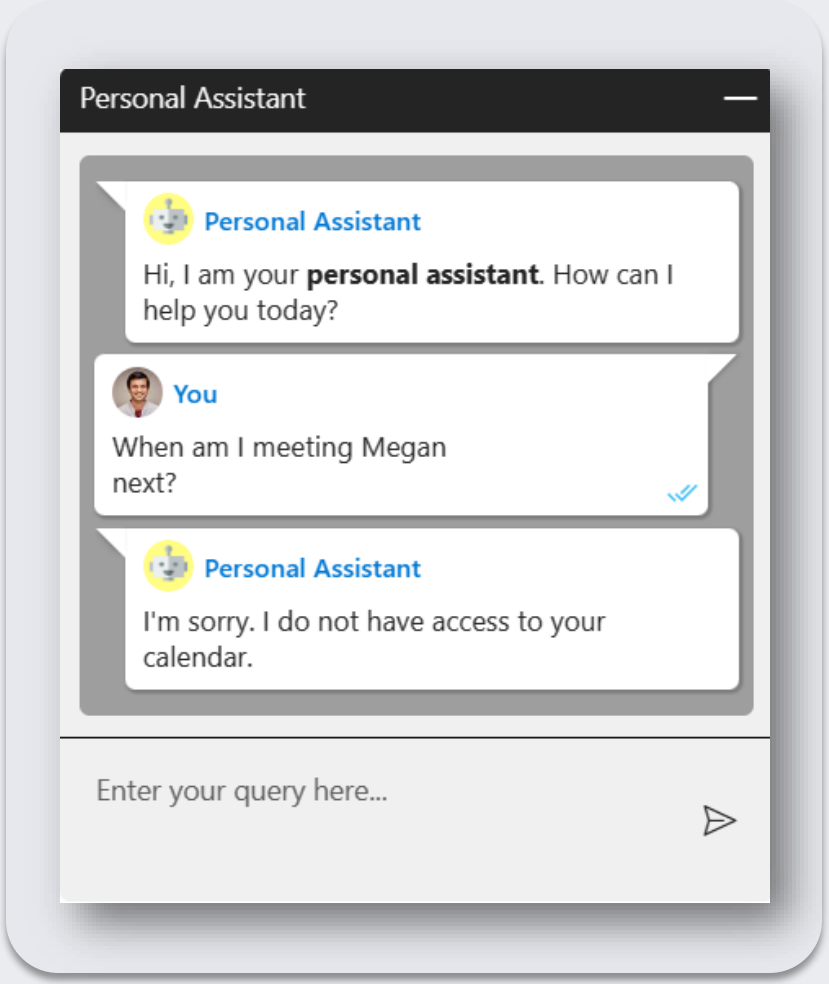
Custom app uses OpenAI API

Call the function
getMyEvents

With argument “futureEvents” as true

OpenAI

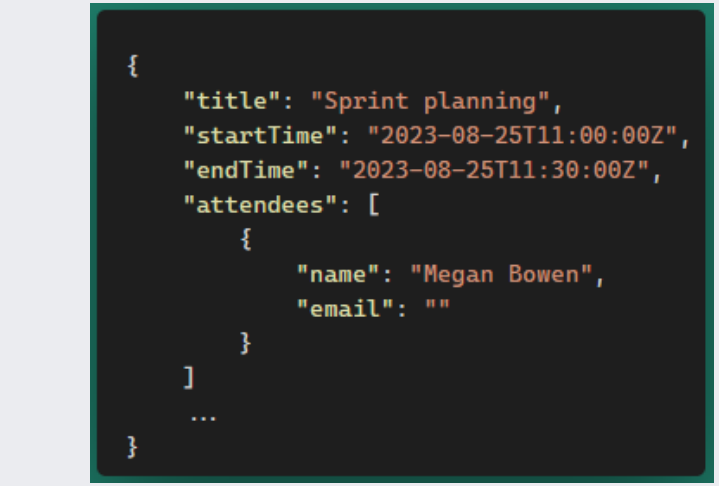
OpenAI Function Calling



Custom app uses OpenAI API

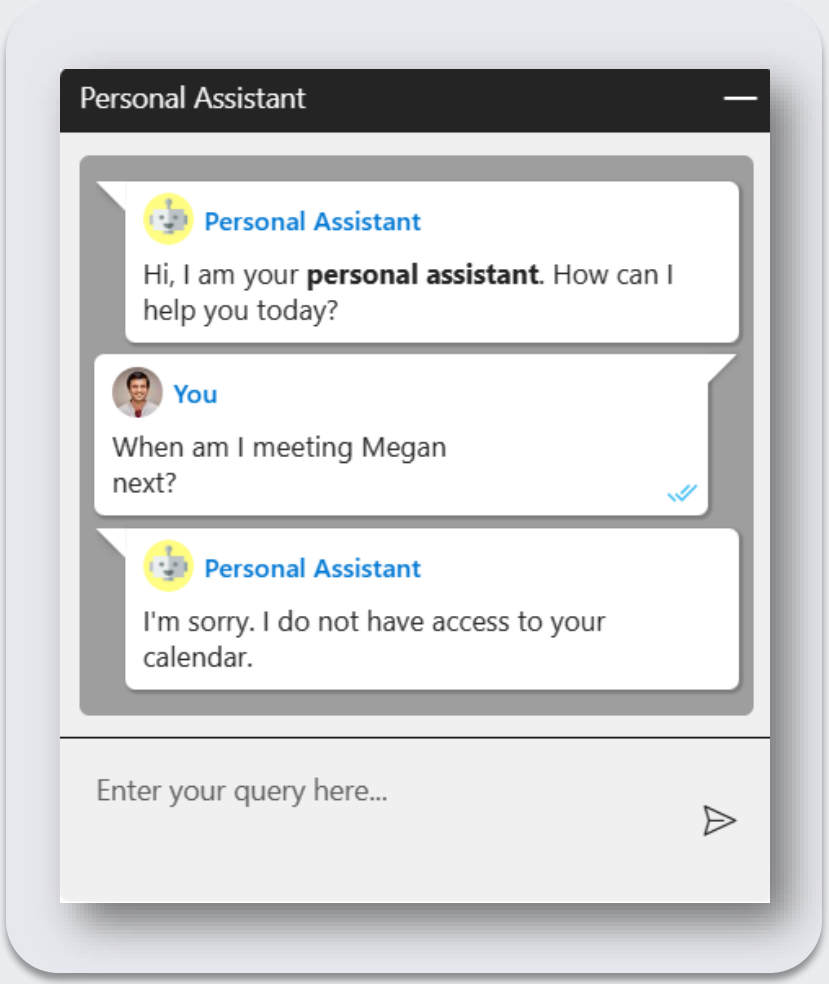
Execute function (which uses Graph)

Pass result to OpenAI



OpenAI

OpenAI Function Calling

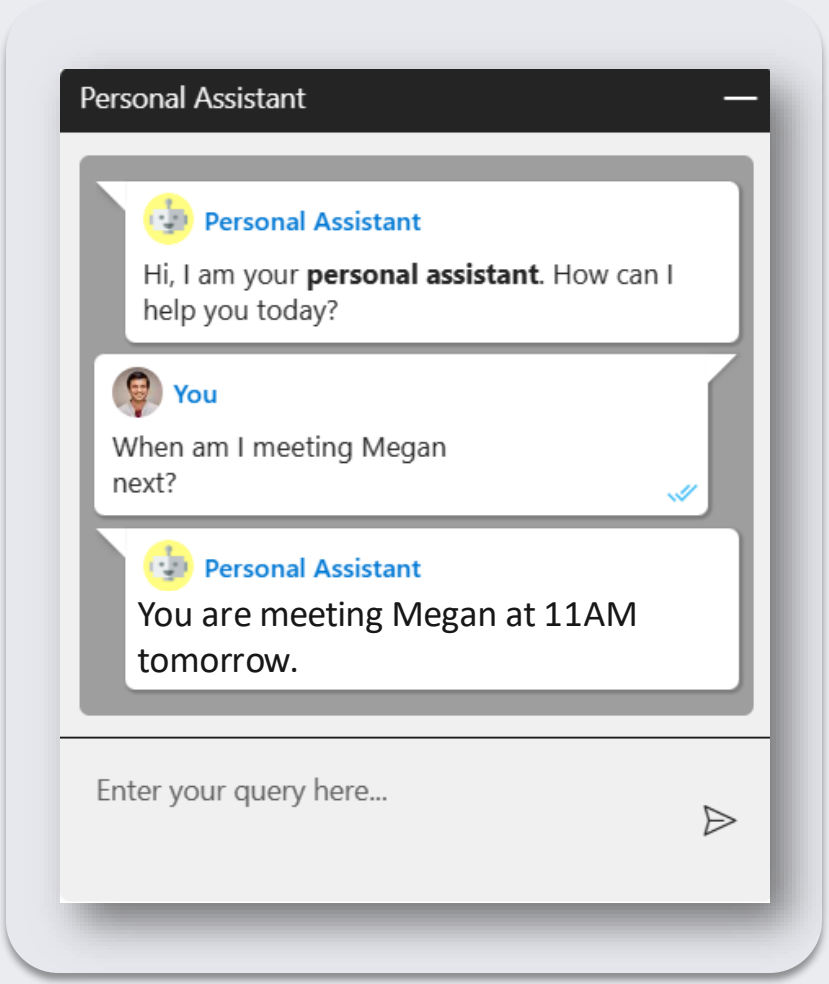


Custom app uses OpenAI API

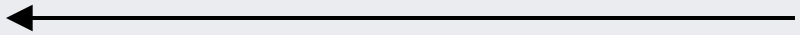
You are meeting Megan at 11AM tomorrow.

OpenAI

OpenAI Function Calling



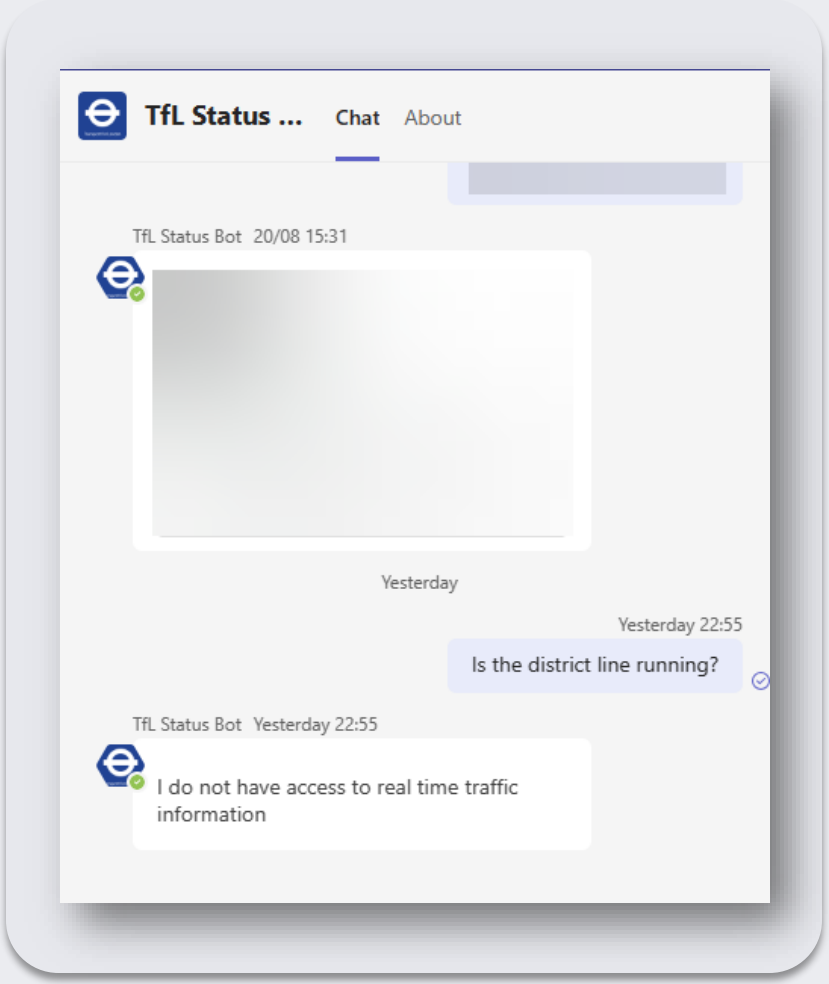
Custom app uses OpenAI API



OpenAI

TfL Teams Bot

OpenAI Function Calling



Teams bot that uses OpenAI API

Is the district line running?

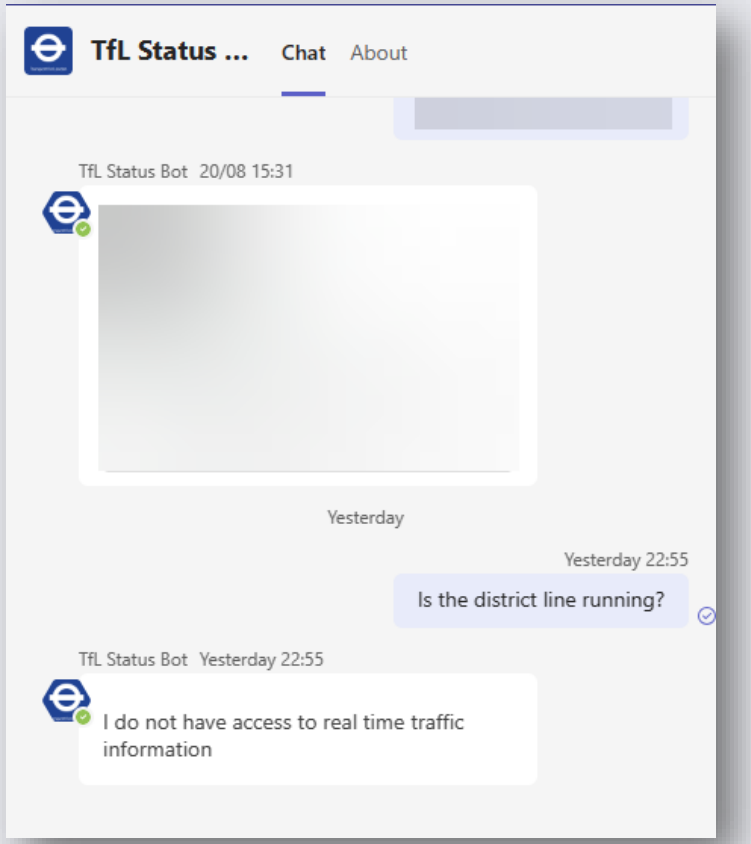
Text definition of

getLineStatus (lineId),
displayLineStatus ()

Along with arguments and their description

OpenAI

OpenAI Function Calling



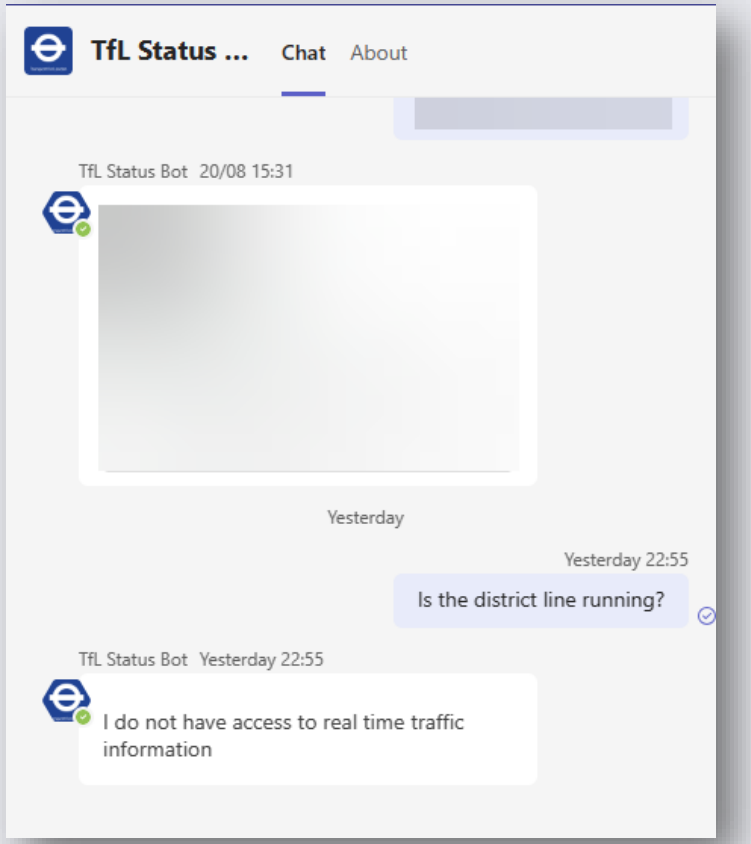
Teams bot that uses OpenAI API

Call the function
getLineStatus

With argument “lineID” as “District”

OpenAI

OpenAI Function Calling



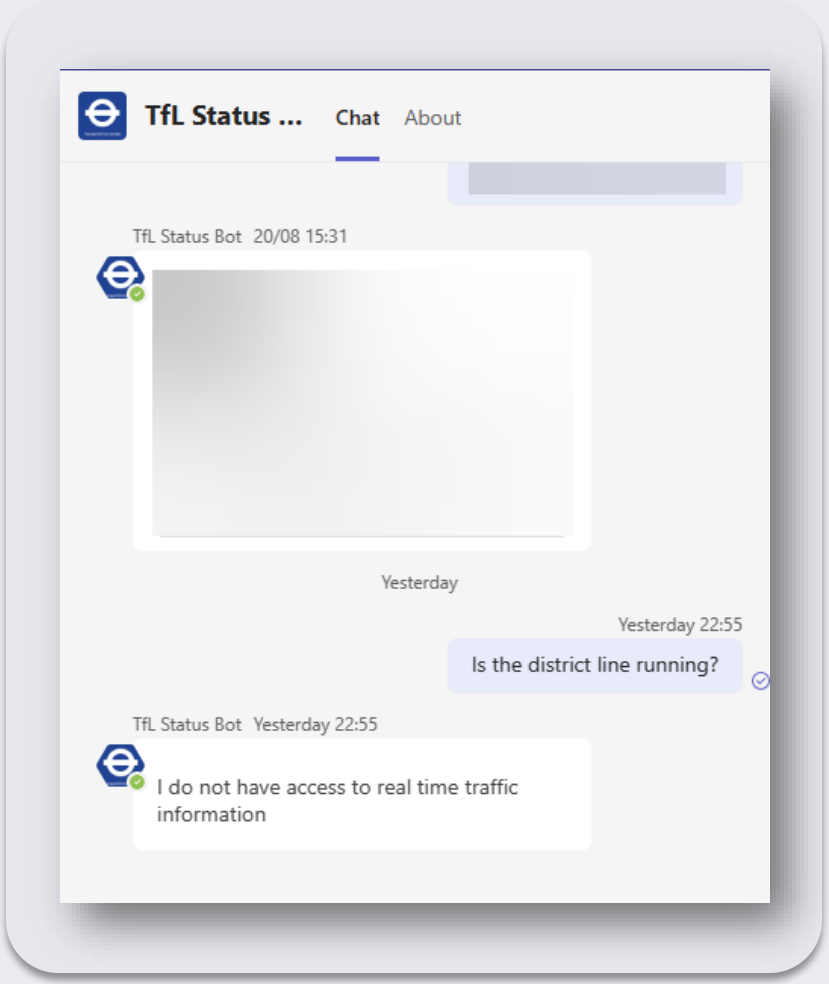
Teams bot that uses OpenAI API

Execute function (which uses TfL API)

Pass result to OpenAI

OpenAI

OpenAI Function Calling

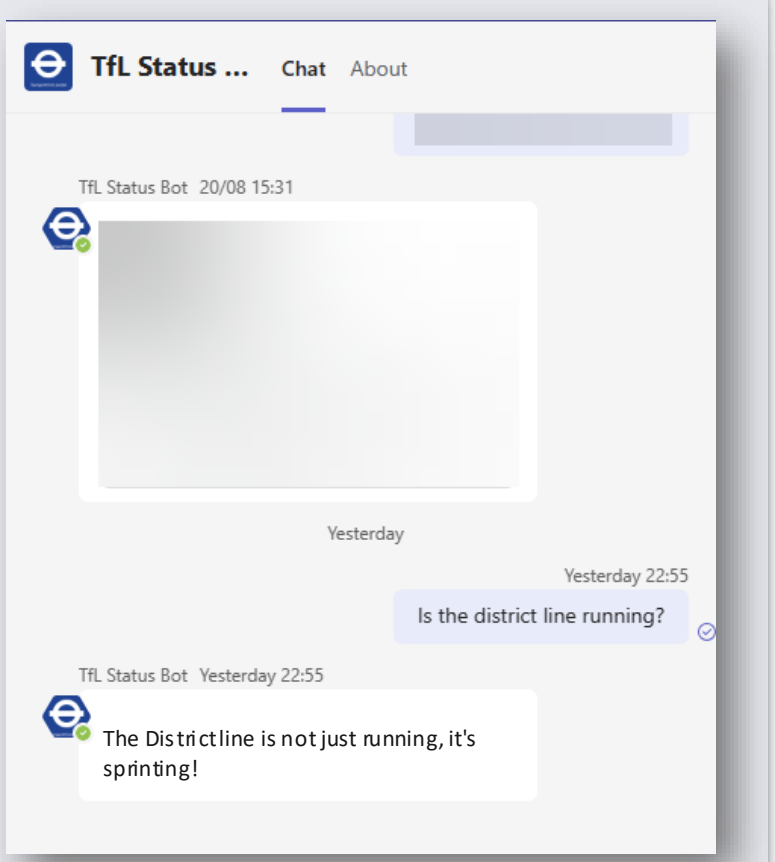


Teams bot that uses OpenAI API

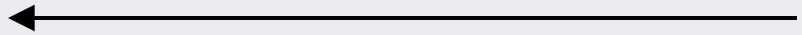
The District line is not just running, it's sprinting!

OpenAI

OpenAI Function Calling



Teams bot that uses OpenAI API



OpenAI

Demo in Postman

Code

Preview README.md

1. Clone this repository

2. Create and populate a `local.settings.json` file in the `source` folder with the following (with your own values):

```
{
"IsEncrypted": false,
"Values": {
    "FUNCTIONS_WORKER_RUNTIME": "node",
    "AzureWebJobsStorage": "",
    "OPENAI_API_KEY": "<YOUR OPENAI API KEY>",
    "GPT_MODELTO_USE": "gpt-4-0613", // might change in the future
    "MicrosoftAppId": "<YOUR MICROSOFT APP REGISTRATION ID>",
    "MicrosoftAppPassword": "<YOUR MICROSOFT APP REGISTRATION CLIENT SECRET>",
    "MicrosoftAppTenantId": "<YOUR MICROSOFT APP REGISTRATION TENANT ID>",
    "MicrosoftAppType": "SingleTenant"
    }
}
```

3. Run the following to install, build and run the code (from the `source` folder):

```
npm install
npm run build
func host start
```

Run on GitHub Codespaces

- Follow the same steps as running locally, but you don't need to run ngrok.
- Once the Azure function is running, you can go into ports tab and make the port `7071` public.
- Then you need to copy the Codespaces URL and use it as the Azure Bot messaging endpoint. e.g. `https://<your-codespace-name>.github.dev/api/messages`

Teams App Manifest

EXPLORER

TEAMS-DEV-SAMPLES [GITHUB]

- bot-openai
- bot-openai-tfl-status
  - .devcontainer
  - assets
  - manifest
  - source
    - .vscode
    - adaptiveCards
    - constants
      - openai.ts
    - helpers
      - openai.ts
      - tfl.ts
    - interfaces
    - messages
    - modules
    - .funcignore
    - host.json
    - package-lock.json
    - package.json
    - tsconfig.json
  - .gitignore
  - README.md
- bot-openai-weather
- bot-proactive-messaging
- bot-sso-azuread
- bot-sso-openai-personal-assistant

OUTLINE

TIMELINE

Preview README.md

1. Clone this repository

2. Create and populate a `local.settings.json` file in the `source` folder with the following (with your own values):

```
{
"IsEncrypted": false,
"Values": {
    "FUNCTIONS_WORKER_RUNTIME": "node",
    "AzureWebJobsStorage": "",
    "OPENAI_API_KEY": "<YOUR OPENAI API KEY>",
    "GPT_MODELTO_USE": "gpt-4-0613", // might change in the future
    "MicrosoftAppId": "<YOUR MICROSOFT APP REGISTRATION ID>",
    "MicrosoftAppPassword": "<YOUR MICROSOFT APP REGISTRATION CLIENT SECRET>",
    "MicrosoftAppTenantId": "<YOUR MICROSOFT APP REGISTRATION TENANT ID>",
    "MicrosoftAppType": "SingleTenant"
    }
}
```

3. Run the following to install, build and run the code (from the `source` folder):

```
npm install
npm run build
func host start
```

Run on GitHub Codespaces

- Follow the same steps as running locally, but you don't need to run ngrok.
- Once the Azure function is running, you can go into ports tab and make the port `7071` public.
- Then you need to copy the Codespaces URL and use it as the Azure Bot messaging endpoint. e.g. `https://<your-codespace-name>.github.dev/api/messages`

Teams App Manifest

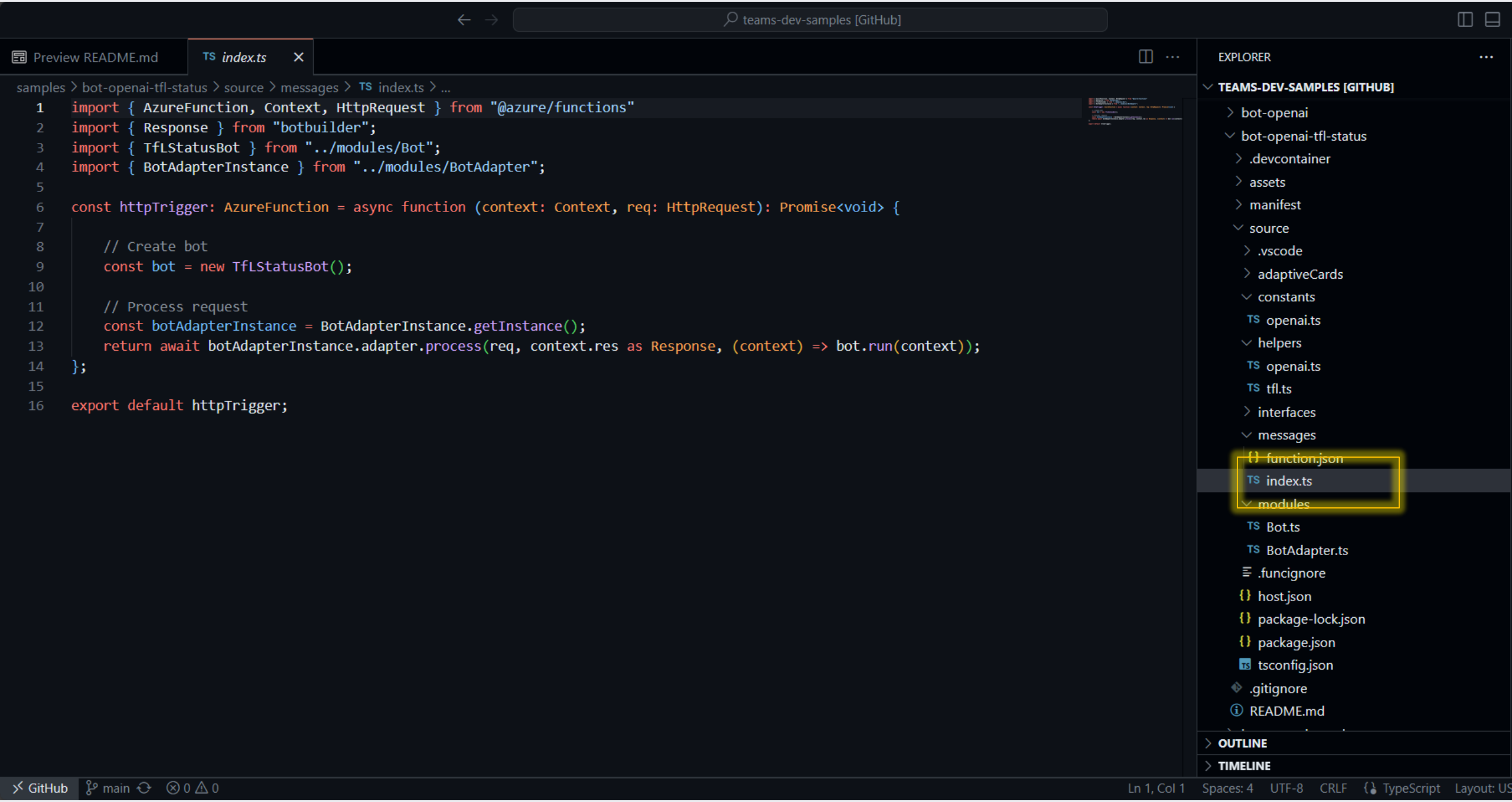
EXPLORER

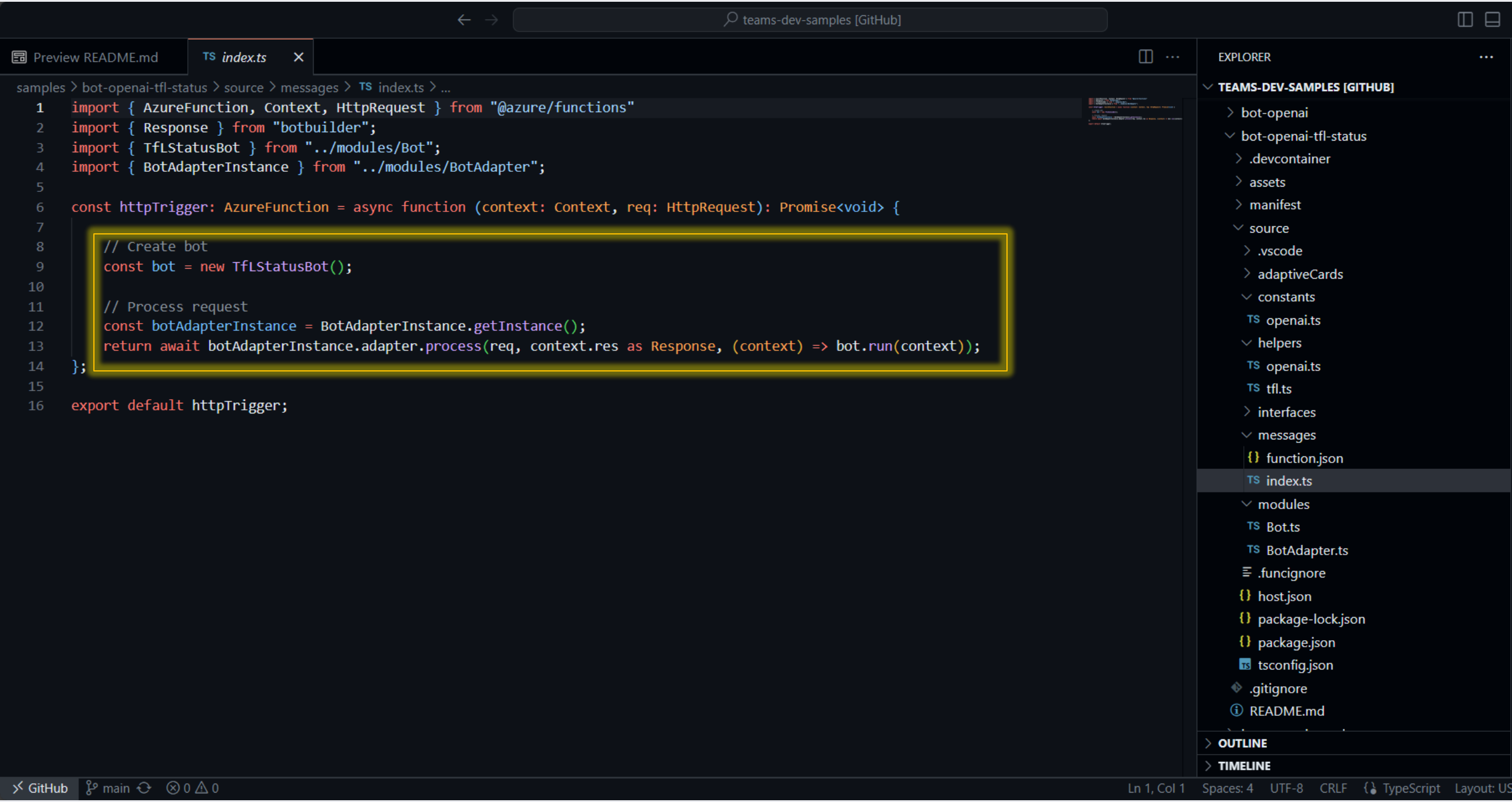
TEAMS-DEV-SAMPLES [GITHUB]

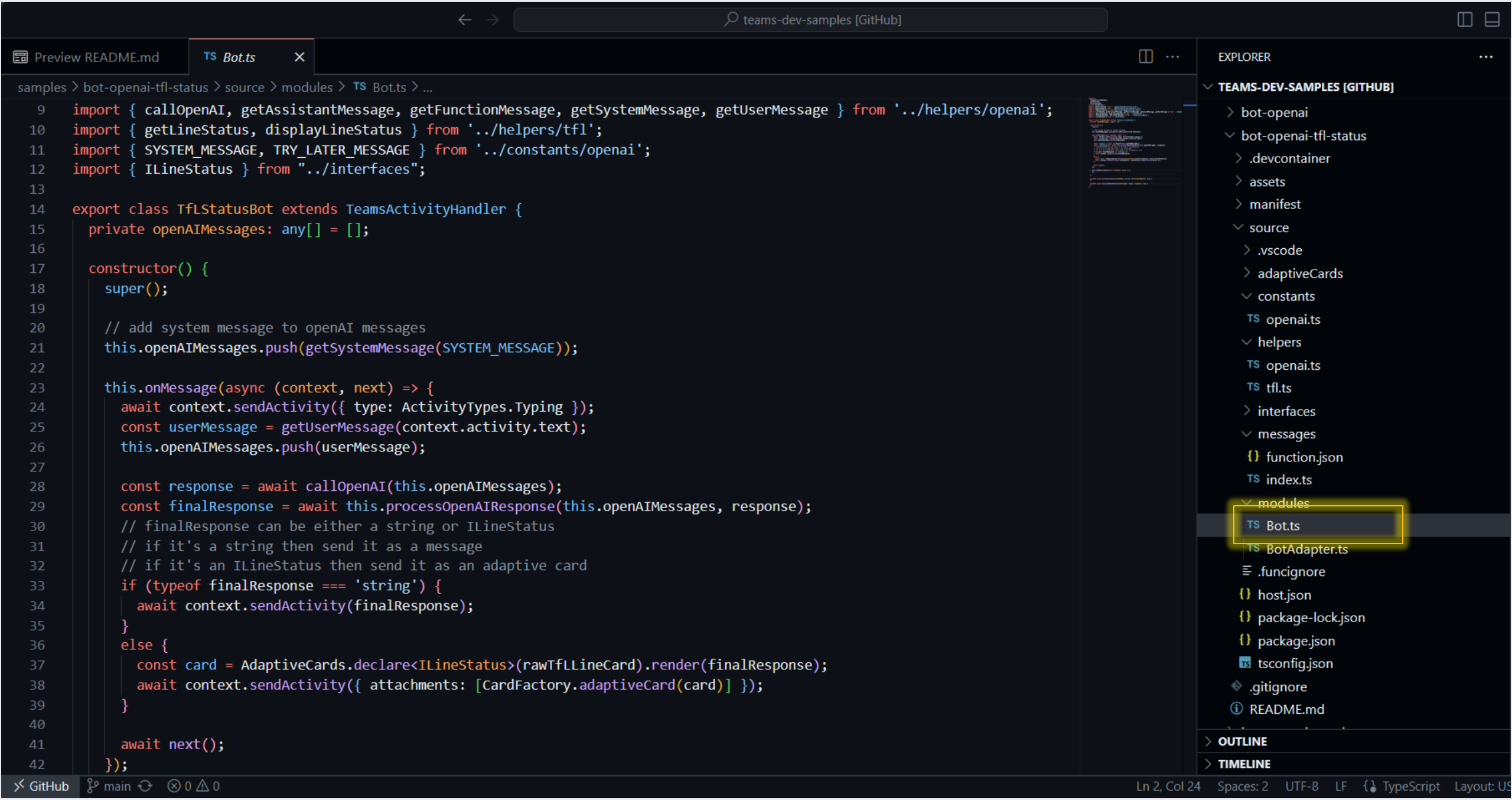
- bot-openai
- bot-openai-tfl-status
  - .devcontainer
  - assets
  - manifest
  - source
    - .vscode
    - adaptiveCards
    - constants
      - openai.ts
    - helpers
      - openai.ts
      - tfl.ts
    - interfaces
    - messages
    - modules
    - .funcignore
    - host.json
    - package-lock.json
    - package.json
    - tsconfig.json
  - .gitignore
  - README.md
- bot-openai-weather
- bot-proactive-messaging
- bot-sso-azuread
- bot-sso-openai-personal-assistant

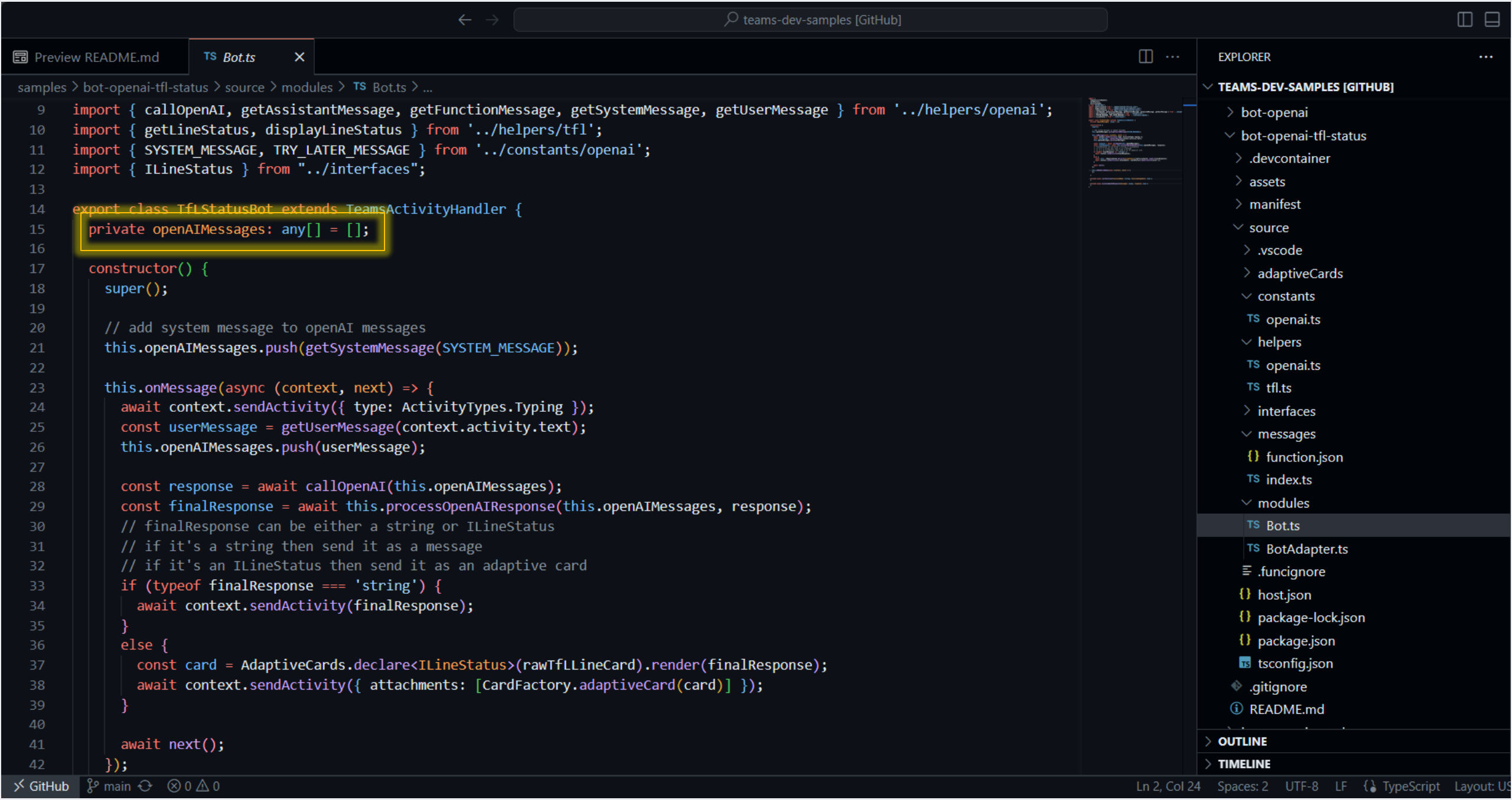
OUTLINE

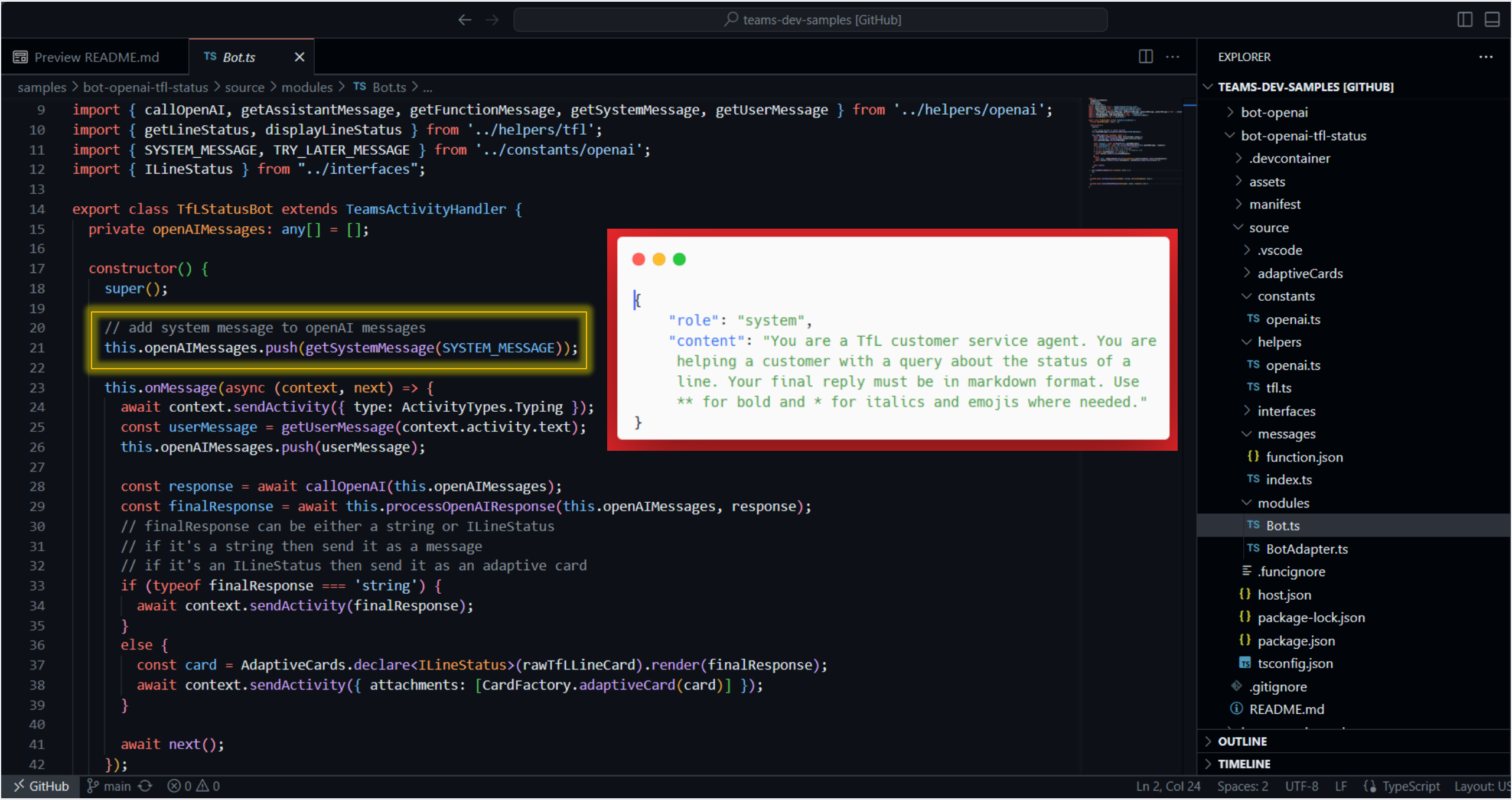
TIMELINE

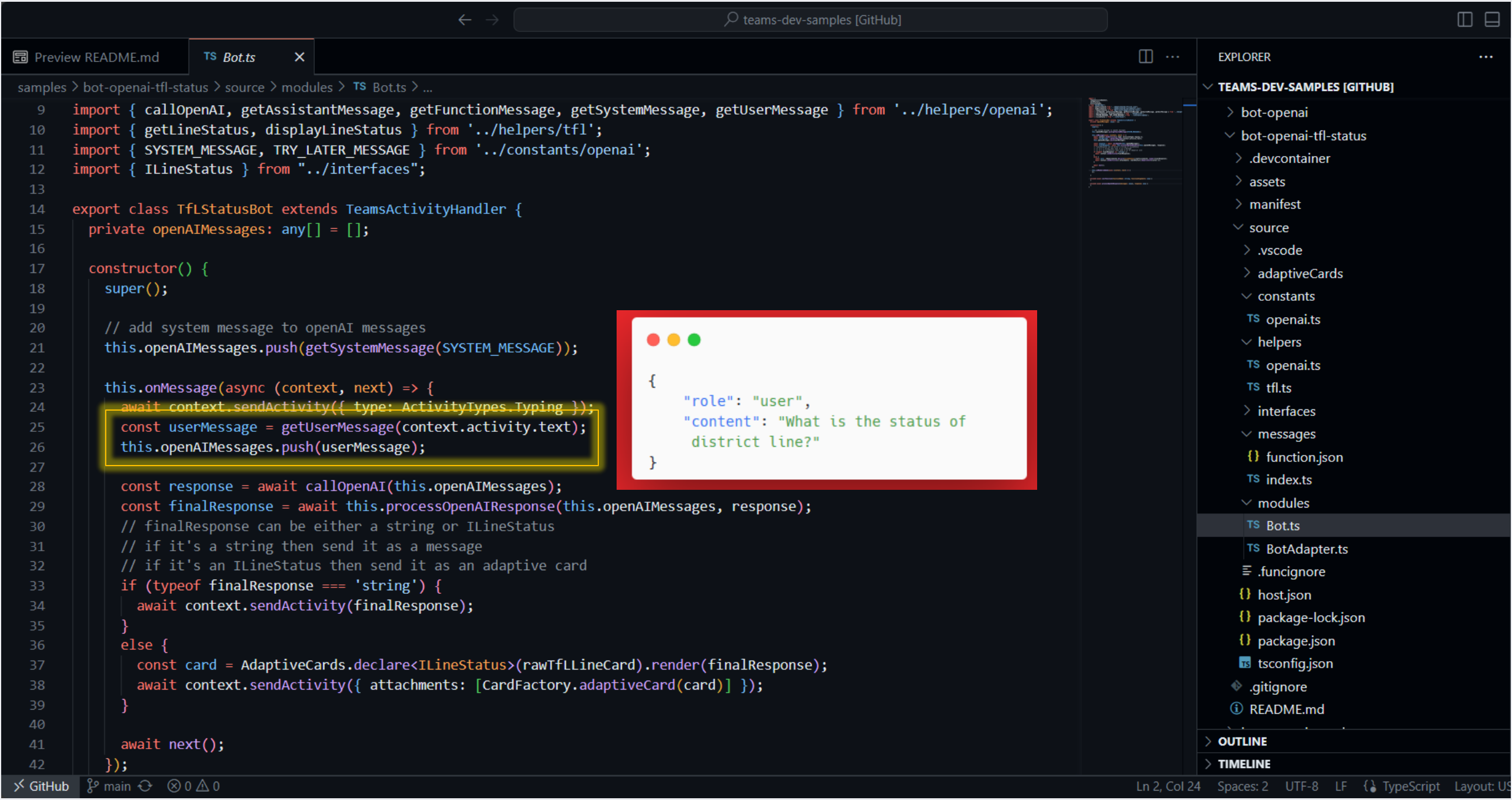

teams-dev-samples [GitHub]
Preview README.md
index.ts
samples > bot-openai-tfl-status > source > messages > index.ts > ...
1 import { AzureFunction, Context, HttpRequest } from "@azure/functions"
2 import { Response } from "botbuilder";
3 import { TfLStatusBot } from "../modules/Bot";
4 import { BotAdapterInstance } from "../modules/BotAdapter";
5
6 const httpTrigger: AzureFunction = async function (context: Context, req: HttpRequest): Promise<void> {
7
8 // Create bot
9 const bot = new TfLStatusBot();
10
11 // Process request
12 const botAdapterInstance = BotAdapterInstance.getInstance();
13 return await botAdapterInstance.adapter.process(req, context.res as Response, (context) => bot.run(context));
14 };
15
16 export default httpTrigger;
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
bot-openai-tfl-status
.devcontainer
assets
manifest
source
.vscode
adaptiveCards
constants
openai.ts
helpers
openai.ts
tfl.ts
interfaces
messages
function.json
index.ts
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
0 0
Ln 1, Col 1
Spaces: 4
UTF-8
CRLF
TypeScript
Layout: US

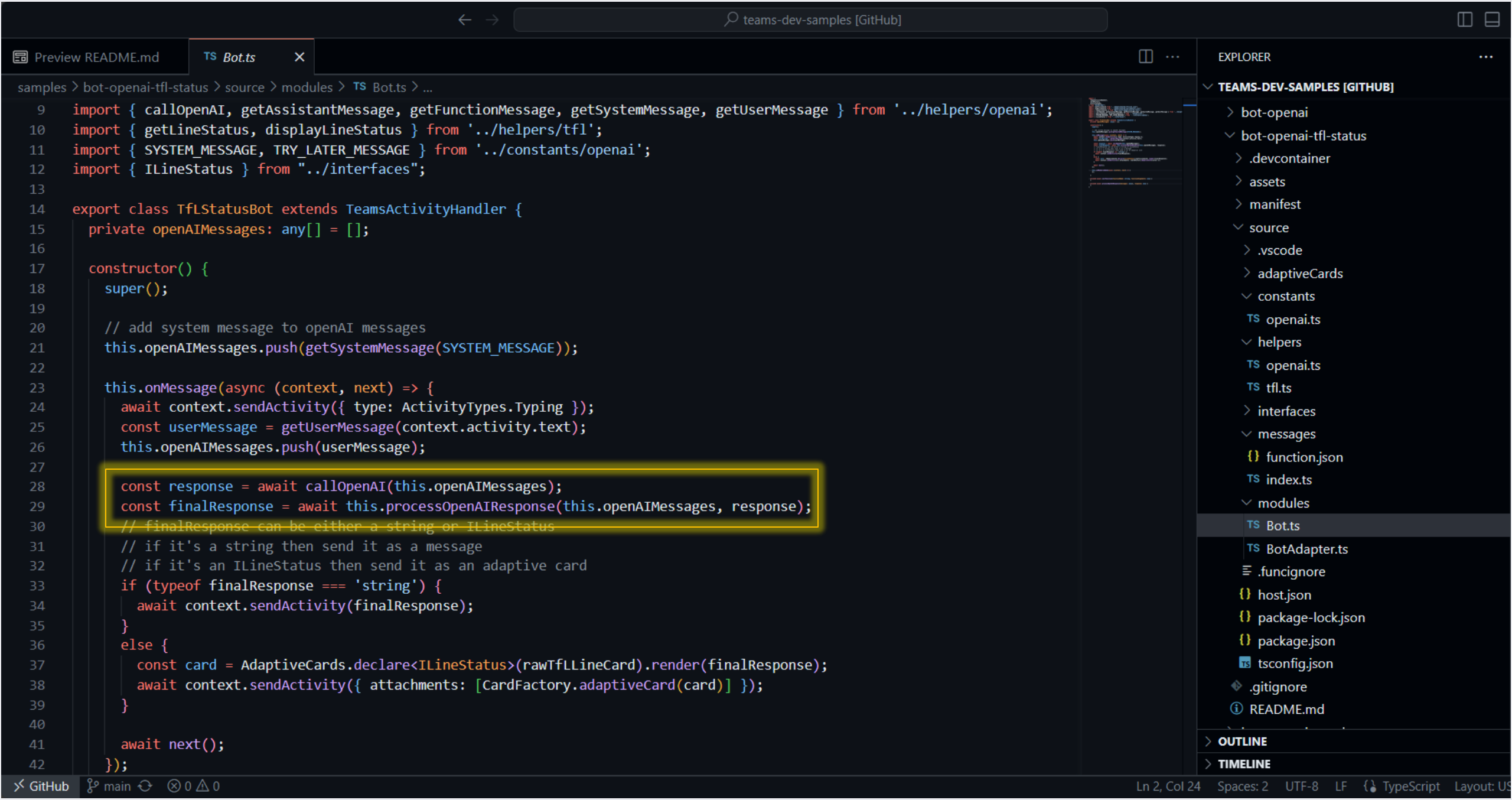

teams-dev-samples [GitHub]
Preview README.md
index.ts
samples > bot-openai-tfl-status > source > messages > index.ts > ...
1 import { AzureFunction, Context, HttpRequest } from "@azure/functions"
2 import { Response } from "botbuilder";
3 import { TfLStatusBot } from "../modules/Bot";
4 import { BotAdapterInstance } from "../modules/BotAdapter";
5
6 const httpTrigger: AzureFunction = async function (context: Context, req: HttpRequest): Promise<void> {
7
8 // Create bot
9 const bot = new TfLStatusBot();
10
11 // Process request
12 const botAdapterInstance = BotAdapterInstance.getInstance();
13 return await botAdapterInstance.adapter.process(req, context.res as Response, (context) => bot.run(context));
14 };
15
16 export default httpTrigger;
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
bot-openai-tfl-status
.devcontainer
assets
manifest
source
.vscode
adaptiveCards
constants
openai.ts
helpers
openai.ts
tfl.ts
interfaces
messages
function.json
index.ts
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
0 0
Ln 1, Col 1
Spaces: 4
UTF-8
CRLF
TypeScript

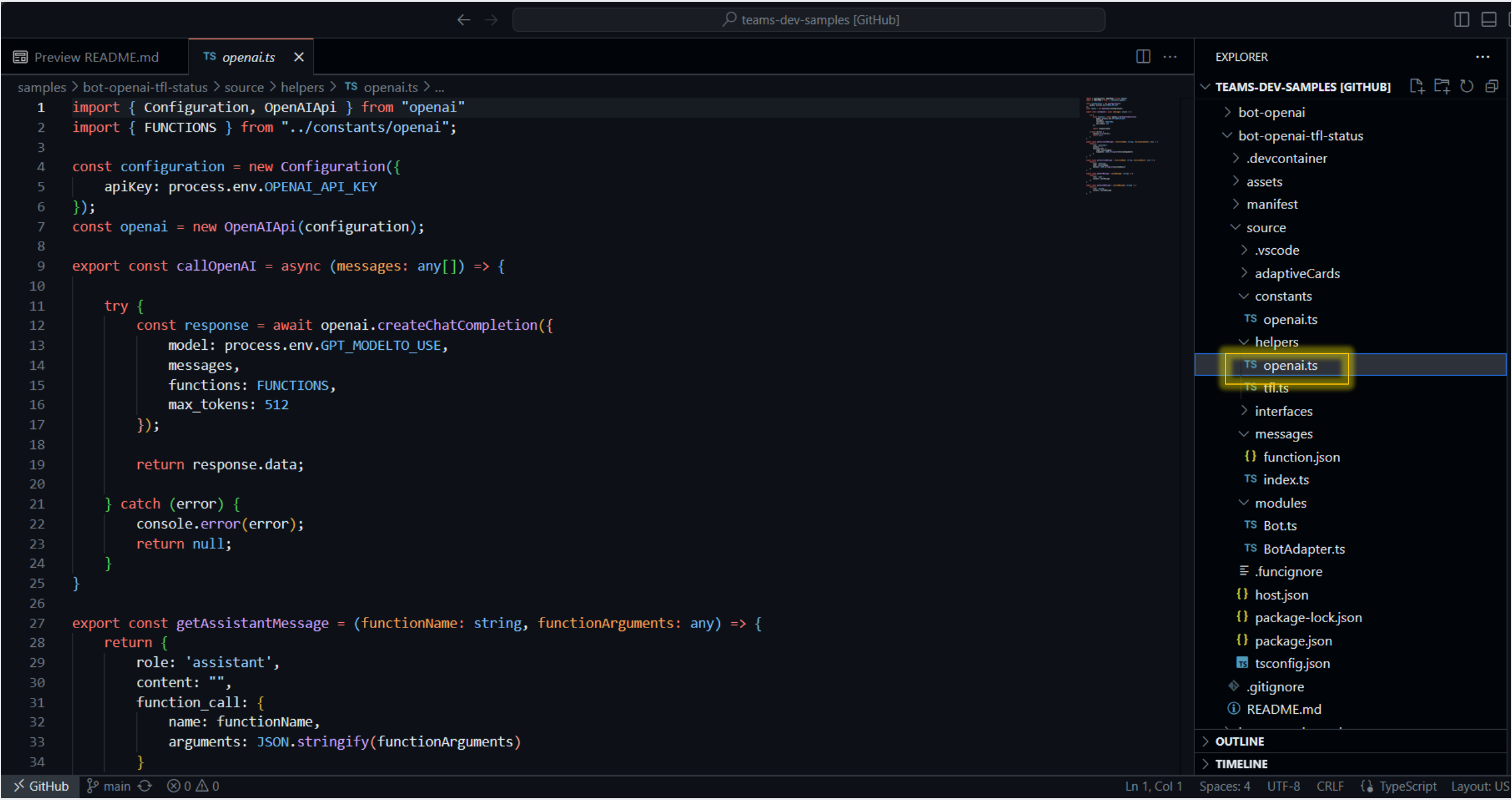

teams-dev-samples [GitHub]
Preview README.md
Bot.ts
samples > bot-openai-tfl-status > source > modules > Bot.ts > ...
import { callOpenAI, getAssistantMessage, getFunctionMessage, getSystemMessage, getUserMessage } from '../helpers/openai';
import { getLineStatus, displayLineStatus } from '../helpers/tfl';
import { SYSTEM_MESSAGE, TRY_LATER_MESSAGE } from '../constants/openai';
import { ILineStatus } from "../interfaces";
export class TfLStatusBot extends TeamsActivityHandler {
private openAIMessages: any[] = [];
constructor() {
super();
// add system message to openAI messages
this.openAIMessages.push(getSystemMessage(SYSTEM_MESSAGE));
this.onMessage(async (context, next) => {
await context.sendActivity({ type: ActivityTypes.Typing });
const userMessage = getUserMessage(context.activity.text);
this.openAIMessages.push(userMessage);
const response = await callOpenAI(this.openAIMessages);
const finalResponse = await this.processOpenAIResponse(this.openAIMessages, response);
// finalResponse can be either a string or ILineStatus
// if it's a string then send it as a message
// if it's an ILineStatus then send it as an adaptive card
if (typeof finalResponse === 'string') {
await context.sendActivity(finalResponse);
}
else {
const card = AdaptiveCards.declare<ILineStatus>(rawTfLLineCard).render(finalResponse);
await context.sendActivity({ attachments: [CardFactory.adaptiveCard(card)] });
}
await next();
});
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
bot-openai-tfl-status
.devcontainer
assets
manifest
source
.vscode
adaptiveCards
constants
openai.ts
helpers
openai.ts
tfl.ts
interfaces
messages
function.json
index.ts
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
Ln 2, Col 24
Spaces: 2
UTF-8
LF
TypeScript

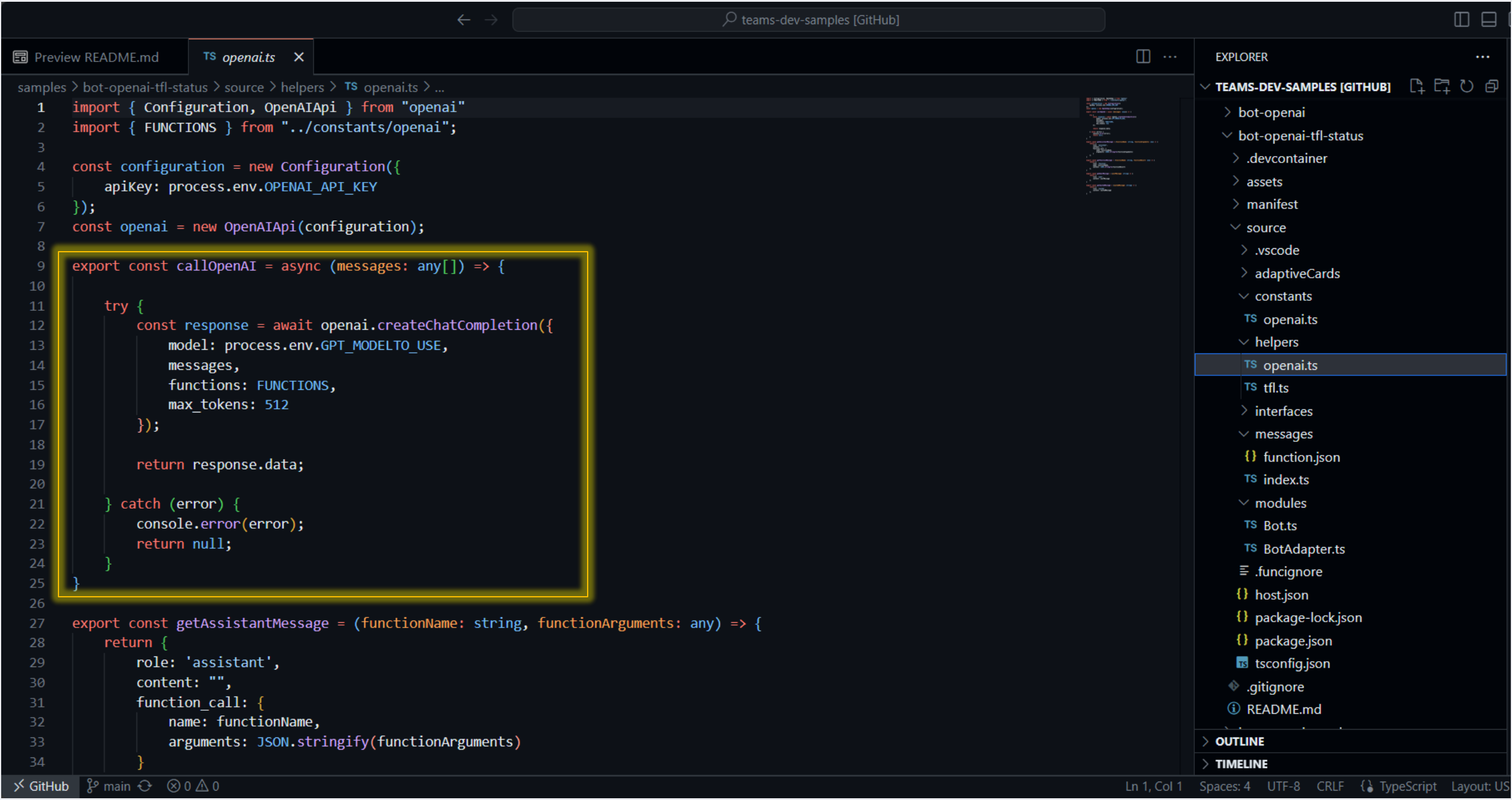

teams-dev-samples [GitHub]
Preview README.md
Bot.ts
samples > bot-openai-tfl-status > source > modules > Bot.ts > ...
import { callOpenAI, getAssistantMessage, getFunctionMessage, getSystemMessage, getUserMessage } from '../helpers/openai';
import { getLineStatus, displayLineStatus } from '../helpers/tfl';
import { SYSTEM_MESSAGE, TRY_LATER_MESSAGE } from '../constants/openai';
import { ILineStatus } from "../interfaces";
export class TflStatusBot extends TeamsActivityHandler {
private openAIMessages: any[] = [];
constructor() {
super();
// add system message to openAI messages
this.openAIMessages.push(getSystemMessage(SYSTEM_MESSAGE));
this.onMessage(async (context, next) => {
await context.sendActivity({ type: ActivityTypes.Typing });
const userMessage = getUserMessage(context.activity.text);
this.openAIMessages.push(userMessage);
const response = await callOpenAI(this.openAIMessages);
const finalResponse = await this.processOpenAIResponse(this.openAIMessages, response);
// finalResponse can be either a string or ILineStatus
// if it's a string then send it as a message
// if it's an ILineStatus then send it as an adaptive card
if (typeof finalResponse === 'string') {
await context.sendActivity(finalResponse);
}
else {
const card = AdaptiveCards.declare<ILineStatus>(rawTfLLineCard).render(finalResponse);
await context.sendActivity({ attachments: [CardFactory.adaptiveCard(card)] });
}
await next();
});
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
bot-openai-tfl-status
.devcontainer
assets
manifest
source
.vscode
adaptiveCards
constants
openai.ts
helpers
openai.ts
tfl.ts
interfaces
messages
function.json
index.ts
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
0 0
Ln 2, Col 24
Spaces: 2
UTF-8
LF
TypeScript

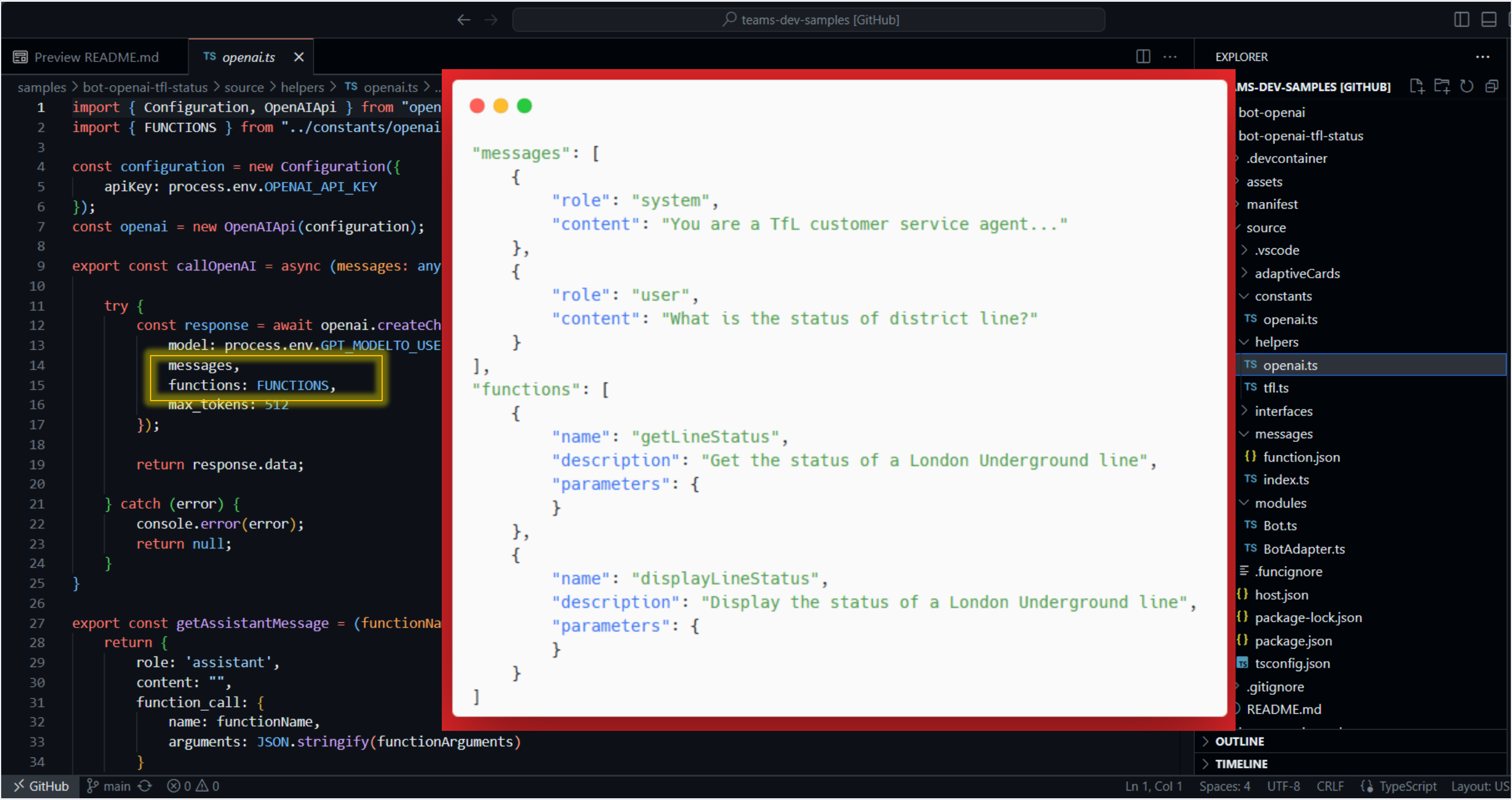

teams-dev-samples [GitHub]
Preview README.md
Bot.ts
samples > bot-openai-tfl-status > source > modules > Bot.ts > ...
import { callOpenAI, getAssistantMessage, getFunctionMessage, getSystemMessage, getUserMessage } from '../helpers/openai';
import { getLineStatus, displayLineStatus } from '../helpers/tfl';
import { SYSTEM_MESSAGE, TRY_LATER_MESSAGE } from '../constants/openai';
import { ILineStatus } from "../interfaces";
export class TfLStatusBot extends TeamsActivityHandler {
private openAIMessages: any[] = [];
constructor() {
super();
// add system message to openAI messages
this.openAIMessages.push(getSystemMessage(SYSTEM_MESSAGE));
this.onMessage(async (context, next) => {
await context.sendActivity({ type: ActivityTypes.Typing });
const userMessage = getUserMessage(context.activity.text);
this.openAIMessages.push(userMessage);
const response = await callOpenAI(this.openAIMessages);
const finalResponse = await this.processOpenAIResponse(this.openAIMessages, response);
// finalResponse can be either a string or ILineStatus
// if it's a string then send it as a message
// if it's an ILineStatus then send it as an adaptive card
if (typeof finalResponse === 'string') {
await context.sendActivity(finalResponse);
}
else {
const card = AdaptiveCards.declare<ILineStatus>(rawTfLLineCard).render(finalResponse);
await context.sendActivity({ attachments: [CardFactory.adaptiveCard(card)] });
}
await next();
});
{
"role": "system",
"content": "You are a TfL customer service agent. You are helping a customer with a query about the status of a line. Your final reply must be in markdown format. Use ** for bold and * for italics and emojis where needed."
}
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
bot-openai-tfl-status
.devcontainer
assets
manifest
source
.vscode
adaptiveCards
constants
openai.ts
helpers
openai.ts
tfl.ts
interfaces
messages
function.json
index.ts
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
Ln 2, Col 24
Spaces: 2
UTF-8
LF
TypeScript

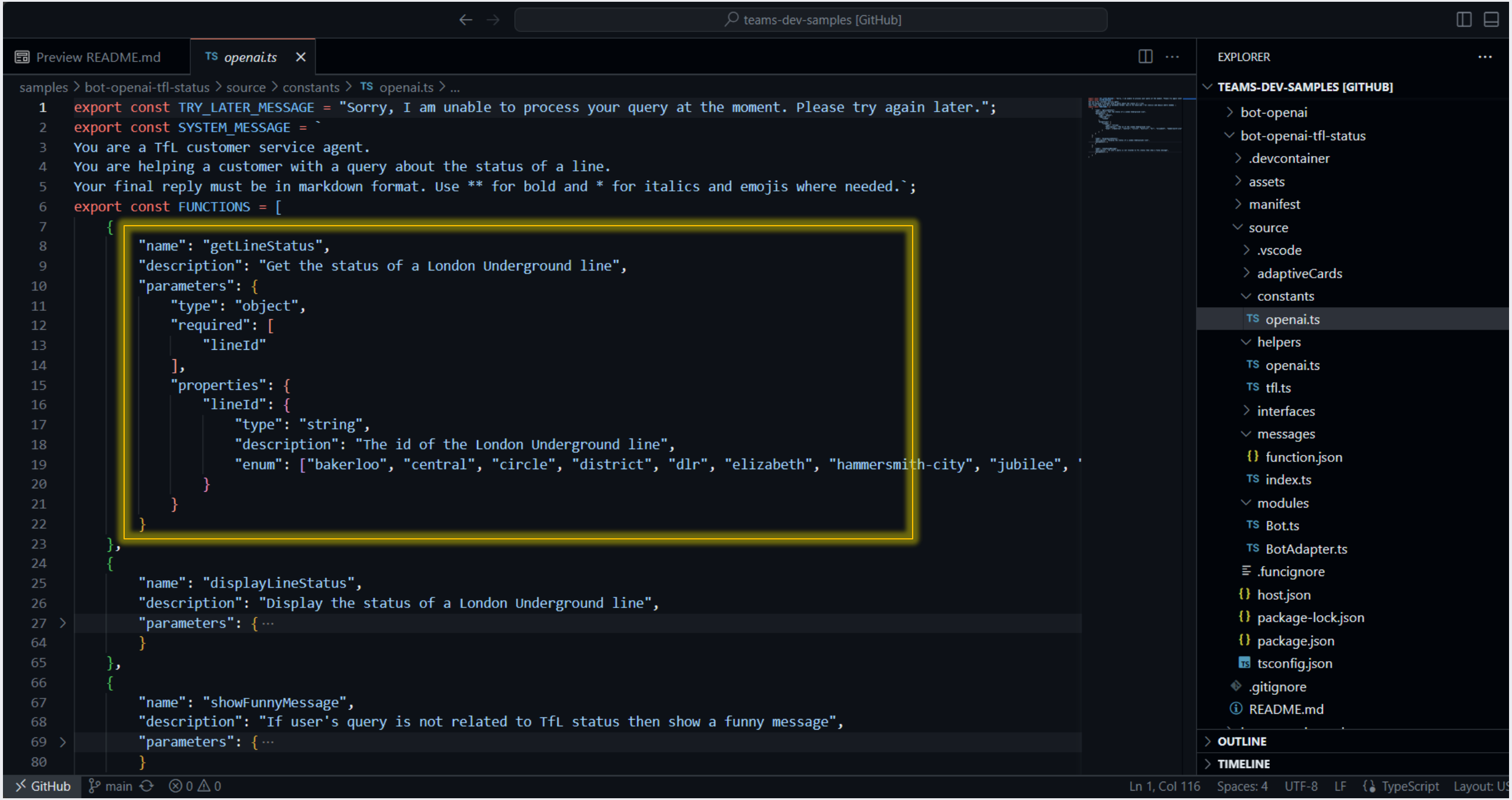

teams-dev-samples [GitHub]
Preview README.md
Bot.ts
samples > bot-openai-tfl-status > source > modules > Bot.ts > ...
import { callOpenAI, getAssistantMessage, getFunctionMessage, getSystemMessage, getUserMessage } from '../helpers/openai';
import { getLineStatus, displayLineStatus } from '../helpers/tfl';
import { SYSTEM_MESSAGE, TRY_LATER_MESSAGE } from '../constants/openai';
import { ILineStatus } from "../interfaces";
export class TfLStatusBot extends TeamsActivityHandler {
private openAIMessages: any[] = [];
constructor() {
super();
// add system message to openAI messages
this.openAIMessages.push(getSystemMessage(SYSTEM_MESSAGE));
this.onMessage(async (context, next) => {
await context.sendActivity({ type: ActivityTypes.Typing });
const userMessage = getUserMessage(context.activity.text);
this.openAIMessages.push(userMessage);
const response = await callOpenAI(this.openAIMessages);
const finalResponse = await this.processOpenAIResponse(this.openAIMessages, response);
// finalResponse can be either a string or ILineStatus
// if it's a string then send it as a message
// if it's an ILineStatus then send it as an adaptive card
if (typeof finalResponse === 'string') {
await context.sendActivity(finalResponse);
}
else {
const card = AdaptiveCards.declare<ILineStatus>(rawTfLLineCard).render(finalResponse);
await context.sendActivity({ attachments: [CardFactory.adaptiveCard(card)] });
}
await next();
});
{
"role": "user",
"content": "What is the status of
district line?"
}
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
bot-openai-tfl-status
.devcontainer
assets
manifest
source
.vscode
adaptiveCards
constants
openai.ts
helpers
openai.ts
tfl.ts
interfaces
messages
function.json
index.ts
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
Ln 2, Col 24
Spaces: 2
UTF-8
LF
TypeScript

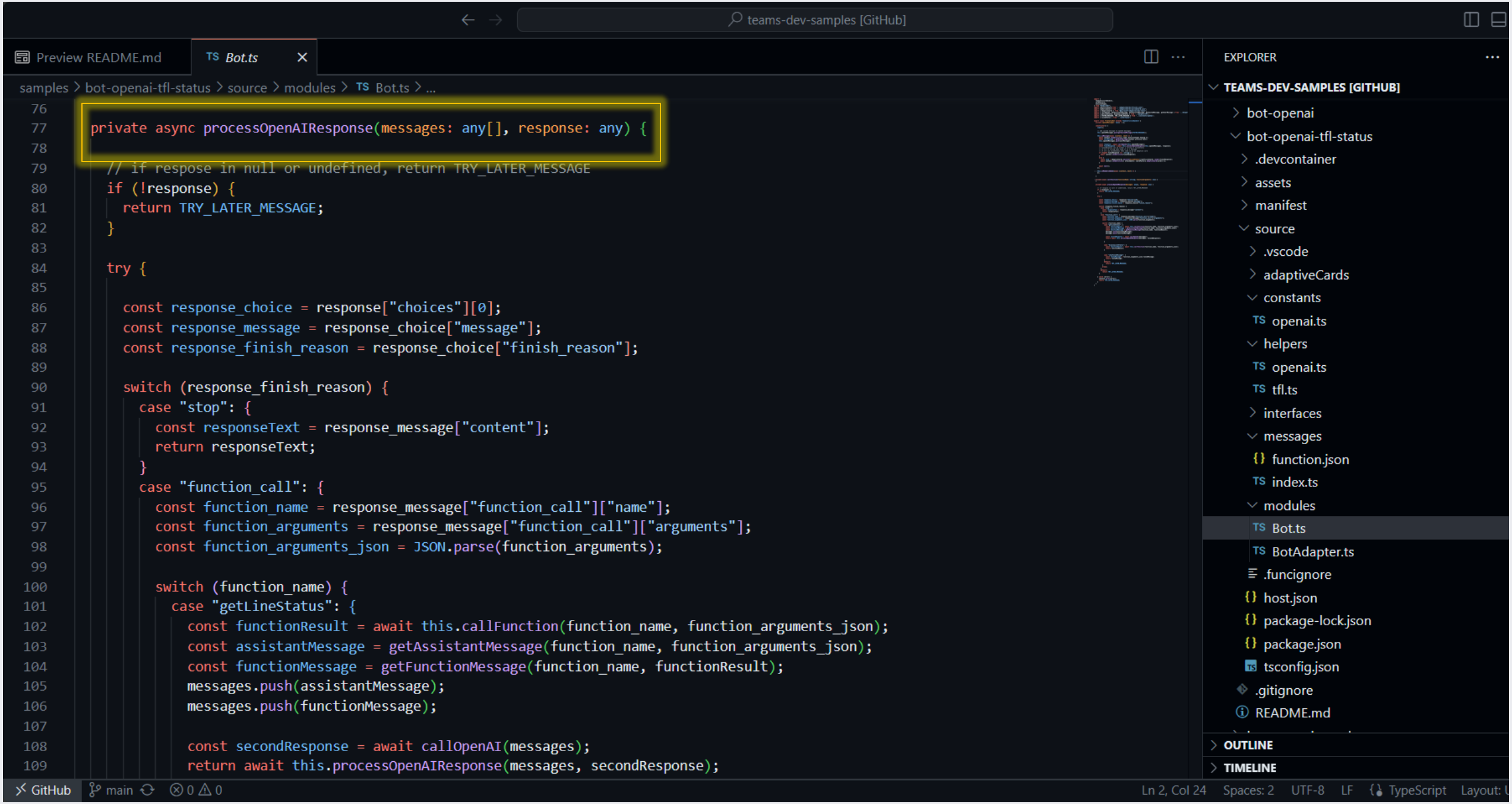

teams-dev-samples [GitHub]
Preview README.md
Bot.ts
samples > bot-openai-tfl-status > source > modules > Bot.ts > ...
import { callOpenAI, getAssistantMessage, getFunctionMessage, getSystemMessage, getUserMessage } from '../helpers/openai';
import { getLineStatus, displayLineStatus } from '../helpers/tfl';
import { SYSTEM_MESSAGE, TRY_LATER_MESSAGE } from '../constants/openai';
import { ILineStatus } from "../interfaces";
export class TfLStatusBot extends TeamsActivityHandler {
private openAIMessages: any[] = [];
constructor() {
super();
// add system message to openAI messages
this.openAIMessages.push(getSystemMessage(SYSTEM_MESSAGE));
this.onMessage(async (context, next) => {
await context.sendActivity({ type: ActivityTypes.Typing });
const userMessage = getUserMessage(context.activity.text);
this.openAIMessages.push(userMessage);
const response = await callOpenAI(this.openAIMessages);
const finalResponse = await this.processOpenAIResponse(this.openAIMessages, response);
// finalResponse can be either a string or ILineStatus
// if it's a string then send it as a message
// if it's an ILineStatus then send it as an adaptive card
if (typeof finalResponse === 'string') {
await context.sendActivity(finalResponse);
}
else {
const card = AdaptiveCards.declare<ILineStatus>(rawTfLLineCard).render(finalResponse);
await context.sendActivity({ attachments: [CardFactory.adaptiveCard(card)] });
}
await next();
});
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
bot-openai-tfl-status
.devcontainer
assets
manifest
source
.vscode
adaptiveCards
constants
openai.ts
helpers
openai.ts
tfl.ts
interfaces
messages
function.json
index.ts
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
Ln 2, Col 24
Spaces: 2
UTF-8
LF
TypeScript

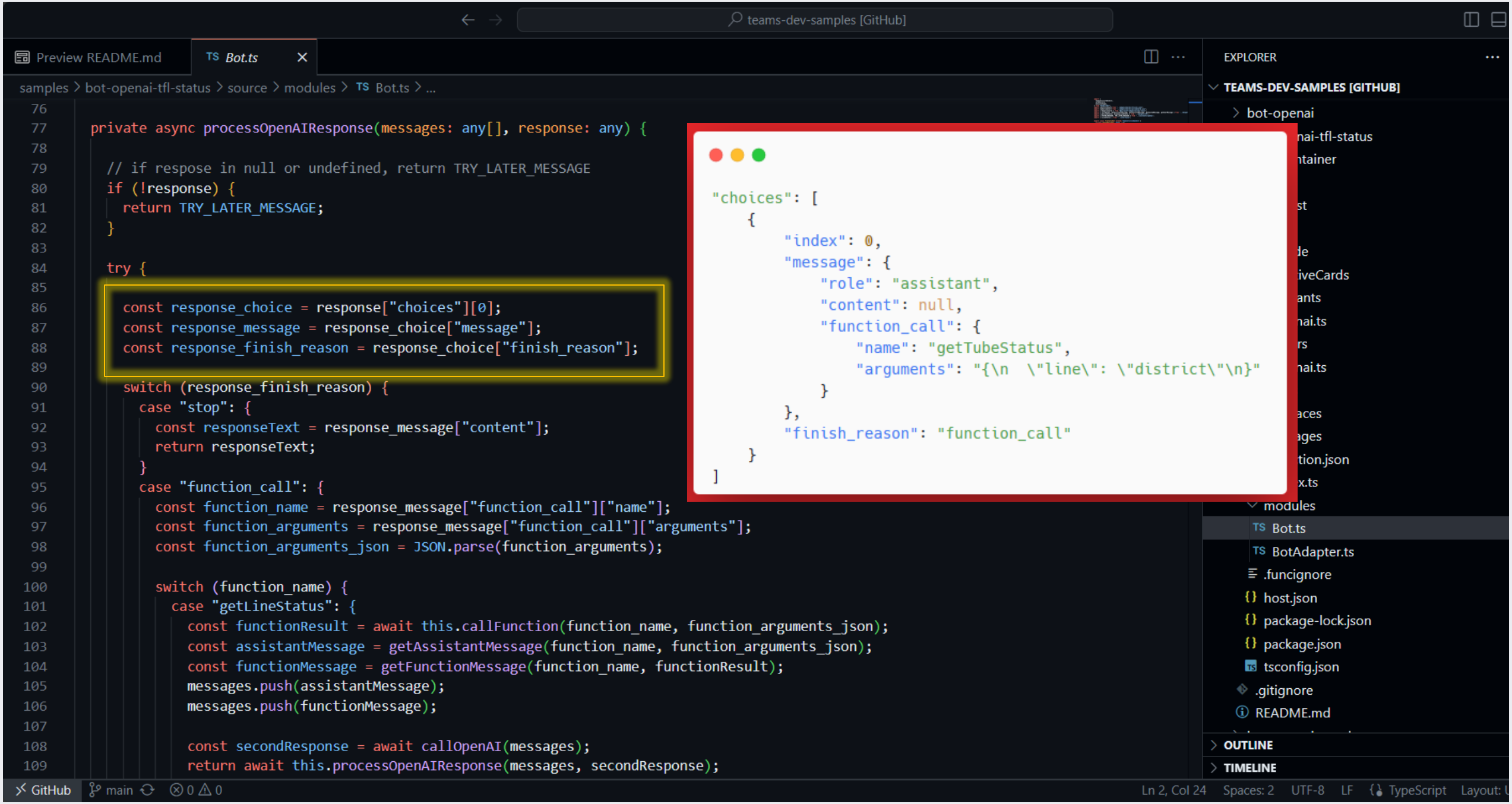

teams-dev-samples [GitHub]
Preview README.md
openai.ts
samples > bot-openai-tfl-status > source > helpers > openai.ts > ...
import { Configuration, OpenAIApi } from "openai"
import { FUNCTIONS } from "../constants/openai";
const configuration = new Configuration({
apiKey: process.env.OPENAI_API_KEY
});
const openai = new OpenAIApi(configuration);
export const callOpenAI = async (messages: any[]) => {
try {
const response = await openai.createChatCompletion({
model: process.env.GPT_MODELTO_USE,
messages,
functions: FUNCTIONS,
max_tokens: 512
});
return response.data;
} catch (error) {
console.error(error);
return null;
}
}
export const getAssistantMessage = (functionName: string, functionArguments: any) => {
return {
role: 'assistant',
content: "",
function_call: {
name: functionName,
arguments: JSON.stringify(functionArguments)
}
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
bot-openai-tfl-status
.devcontainer
assets
manifest
source
.vscode
adaptiveCards
constants
openai.ts
helpers
openai.ts
tfl.ts
interfaces
messages
function.json
index.ts
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
0 0
Ln 1, Col 1
Spaces: 4
UTF-8
CRLF
TypeScript
Layout: US

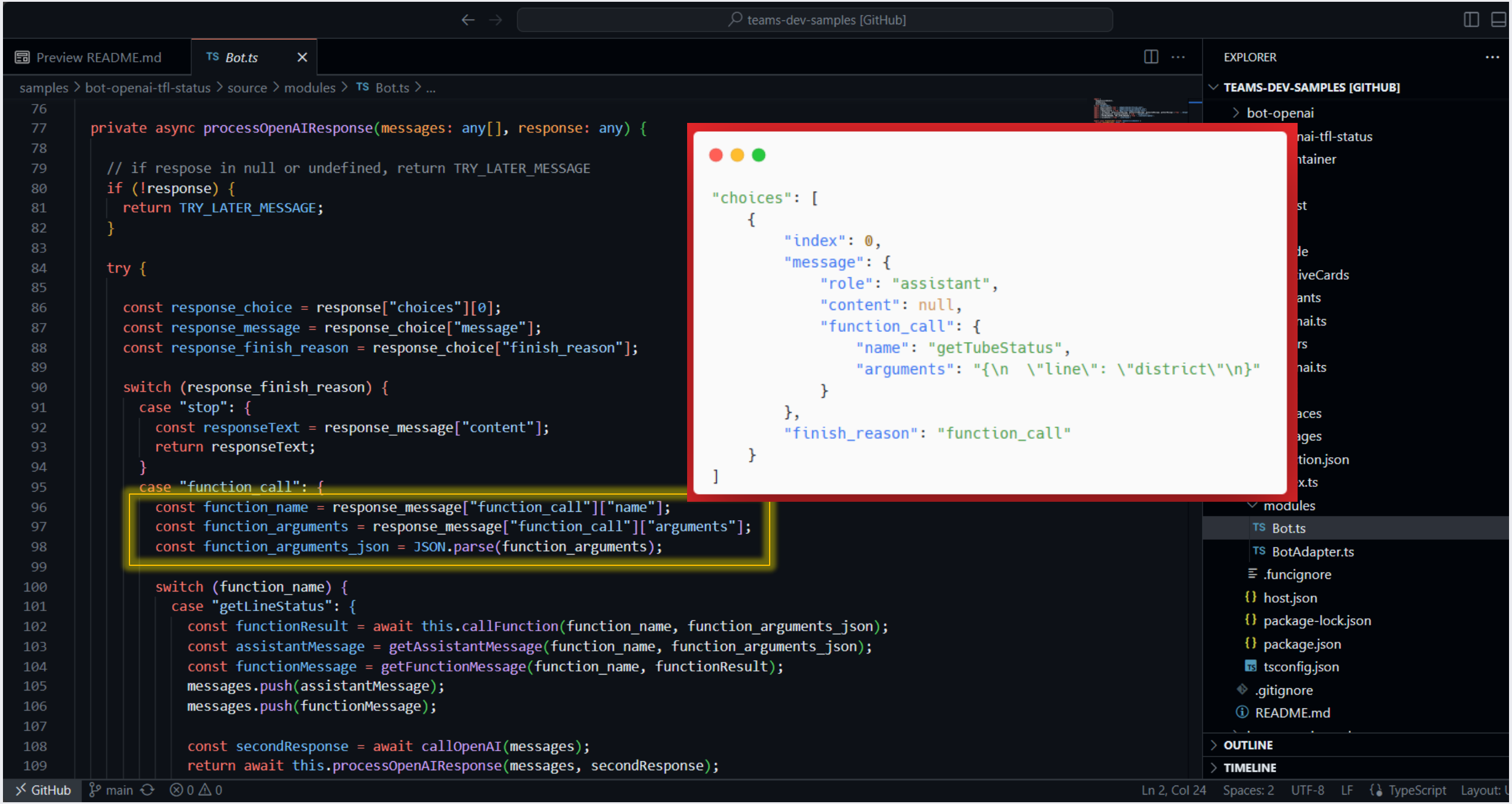

teams-dev-samples [GitHub]
Preview README.md
openai.ts
samples > bot-openai-tfl-status > source > helpers > openai.ts > ...
import { Configuration, OpenAIApi } from "openai"
import { FUNCTIONS } from "../constants/openai";
const configuration = new Configuration({
apiKey: process.env.OPENAI_API_KEY
});
const openai = new OpenAIApi(configuration);
export const callOpenAI = async (messages: any[]) => {
try {
const response = await openai.createChatCompletion({
model: process.env.GPT_MODELTO_USE,
messages,
functions: FUNCTIONS,
max_tokens: 512
});
return response.data;
} catch (error) {
console.error(error);
return null;
}
}
export const getAssistantMessage = (functionName: string, functionArguments: any) => {
return {
role: 'assistant',
content: "",
function_call: {
name: functionName,
arguments: JSON.stringify(functionArguments)
}
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
bot-openai-tfl-status
.devcontainer
assets
manifest
source
.vscode
adaptiveCards
constants
openai.ts
helpers
openai.ts
tfl.ts
interfaces
messages
function.json
index.ts
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
Ln 1, Col 1
Spaces: 4
UTF-8
CRLF
TypeScript

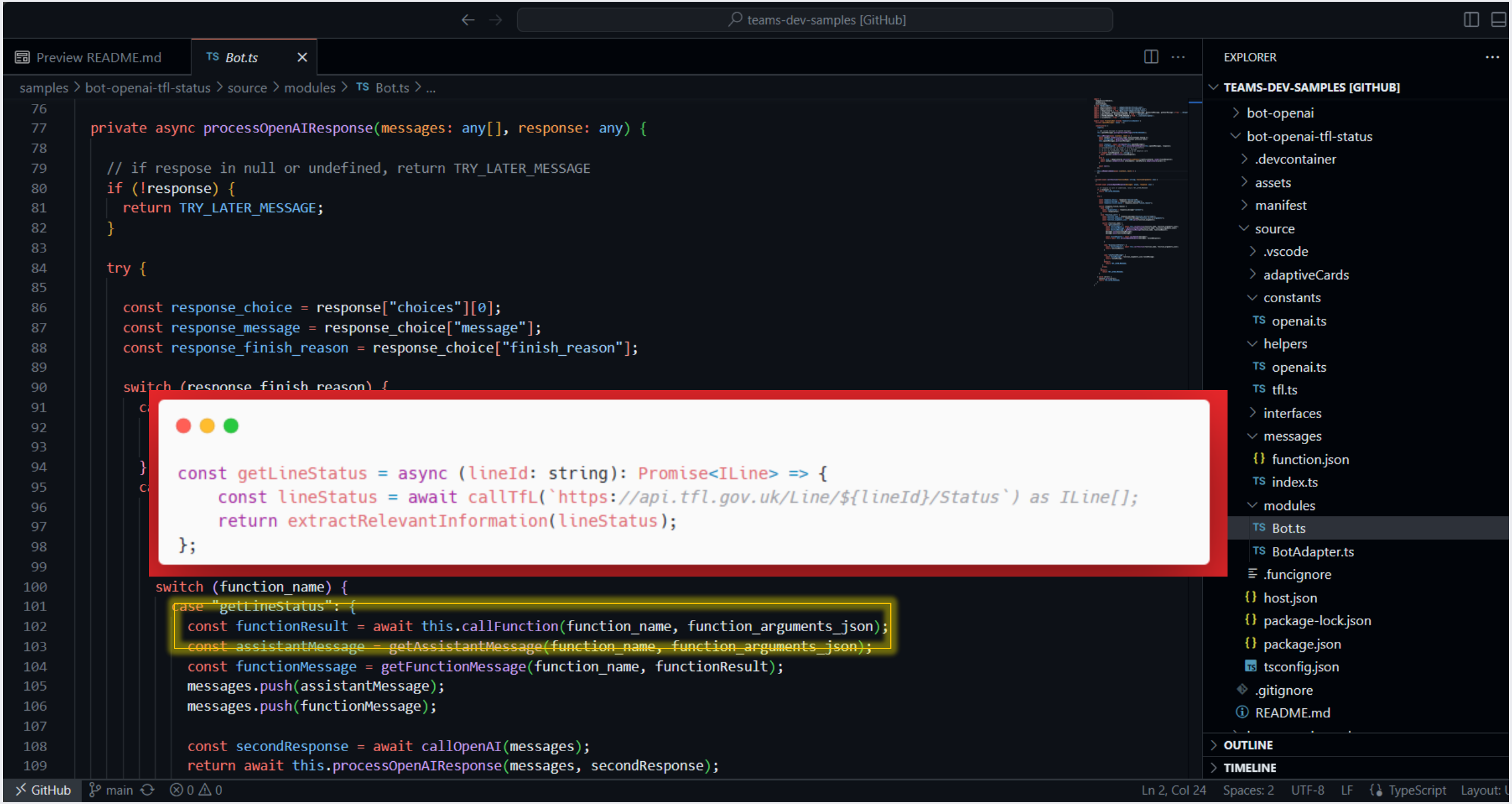

teams-dev-samples [GitHub]
Preview README.md
openai.ts
samples > bot-openai-tfl-status > source > helpers > openai.ts
import { Configuration, OpenAIApi } from "open
import { FUNCTIONS } from "../constants/openai
const configuration = new Configuration({
apiKey: process.env.OPENAI_API_KEY
});
const openai = new OpenAIApi(configuration);
export const callOpenAI = async (messages: any
try {
const response = await openai.createCh
model: process.env.GPT_MODELTO_USE
messages,
functions: FUNCTIONS,
max_tokens: 512
});
return response.data;
} catch (error) {
console.error(error);
return null;
}
}
export const getAssistantMessage = (functionNa
return {
role: 'assistant',
content: "",
function_call: {
name: functionName,
arguments: JSON.stringify(functionArguments)
}
"messages": [
{
"role": "system",
"content": "You are a TfL customer service agent..."
},
{
"role": "user",
"content": "What is the status of district line?"
}
],
"functions": [
{
"name": "getLineStatus",
"description": "Get the status of a London Underground line",
"parameters": {
}
},
{
"name": "displayLineStatus",
"description": "Display the status of a London Underground line",
"parameters": {
}
}
]
EXPLORER
MS-DEV-SAMPLES [GITHUB]
bot-openai
bot-openai-tfl-status
.devcontainer
assets
manifest
source
.vscode
adaptiveCards
constants
openai.ts
helpers
openai.ts
tfl.ts
interfaces
messages
function.json
index.ts
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
Ln 1, Col 1
Spaces: 4
UTF-8
CRLF
TypeScript
Layout: US

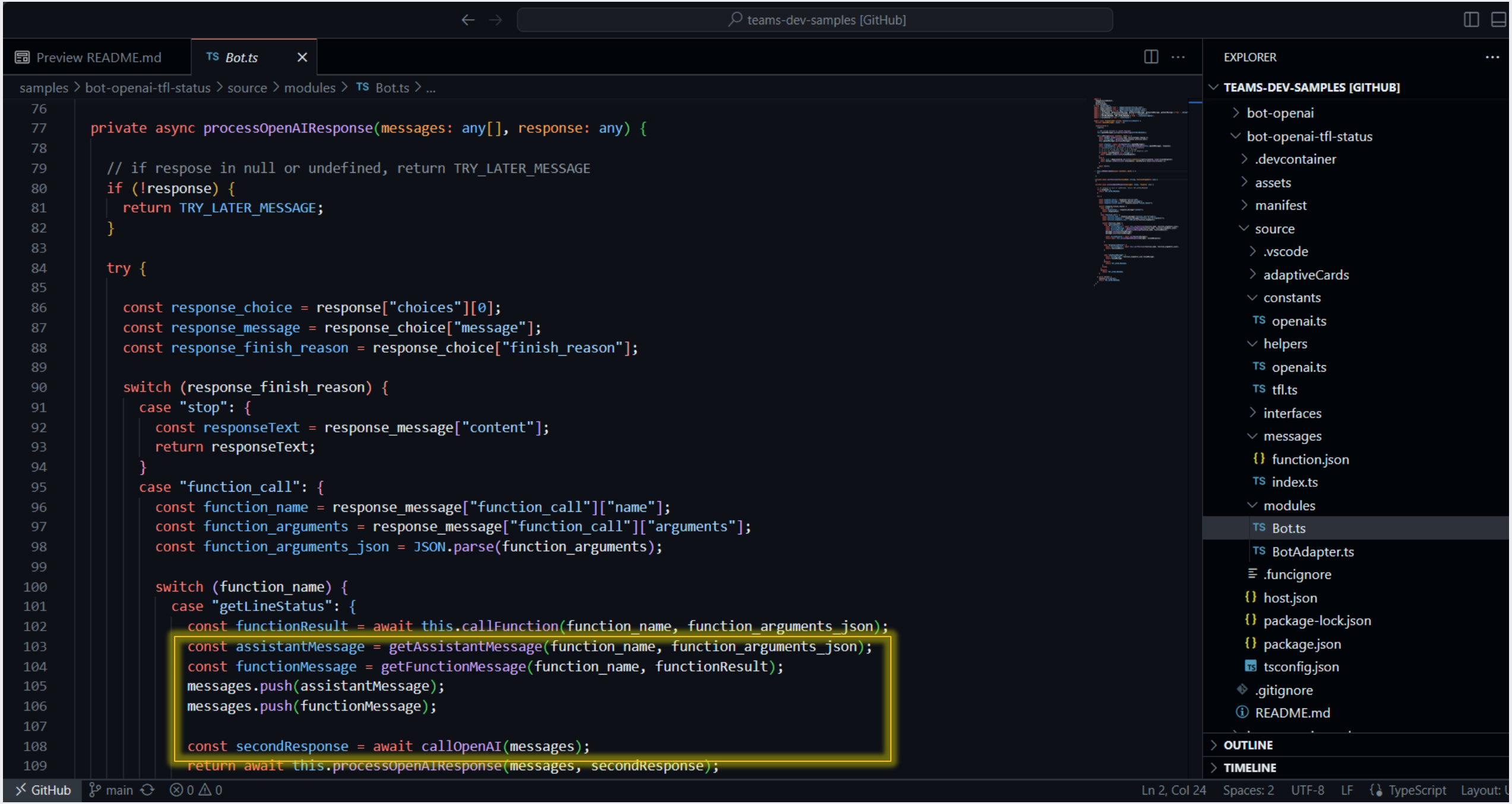

teams-dev-samples [GitHub]
Preview README.md
openai.ts
samples > bot-openai-tfl-status > source > constants > openai.ts > ...
export const TRY_LATER_MESSAGE = "Sorry, I am unable to process your query at the moment. Please try again later.";
export const SYSTEM_MESSAGE = `
You are a TfL customer service agent.
You are helping a customer with a query about the status of a line.
Your final reply must be in markdown format. Use ** for bold and * for italics and emojis where needed.`;
export const FUNCTIONS = [
{
"name": "getLineStatus",
"description": "Get the status of a London Underground line",
"parameters": {
"type": "object",
"required": [
"lineId"
],
"properties": {
"lineId": {
"type": "string",
"description": "The id of the London Underground line",
"enum": ["bakerloo", "central", "circle", "district", "dlr", "elizabeth", "hammersmith-city", "jubilee",
}
}
}
},
{
"name": "displayLineStatus",
"description": "Display the status of a London Underground line",
"parameters": {
}
},
{
"name": "showFunnyMessage",
"description": "If user's query is not related to TfL status then show a funny message",
"parameters": {
}
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
bot-openai-tfl-status
.devcontainer
assets
manifest
source
.vscode
adaptiveCards
constants
openai.ts
helpers
openai.ts
tfl.ts
interfaces
messages
function.json
index.ts
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
0 0
Ln 1, Col 116
Spaces: 4
UTF-8
LF
TypeScript

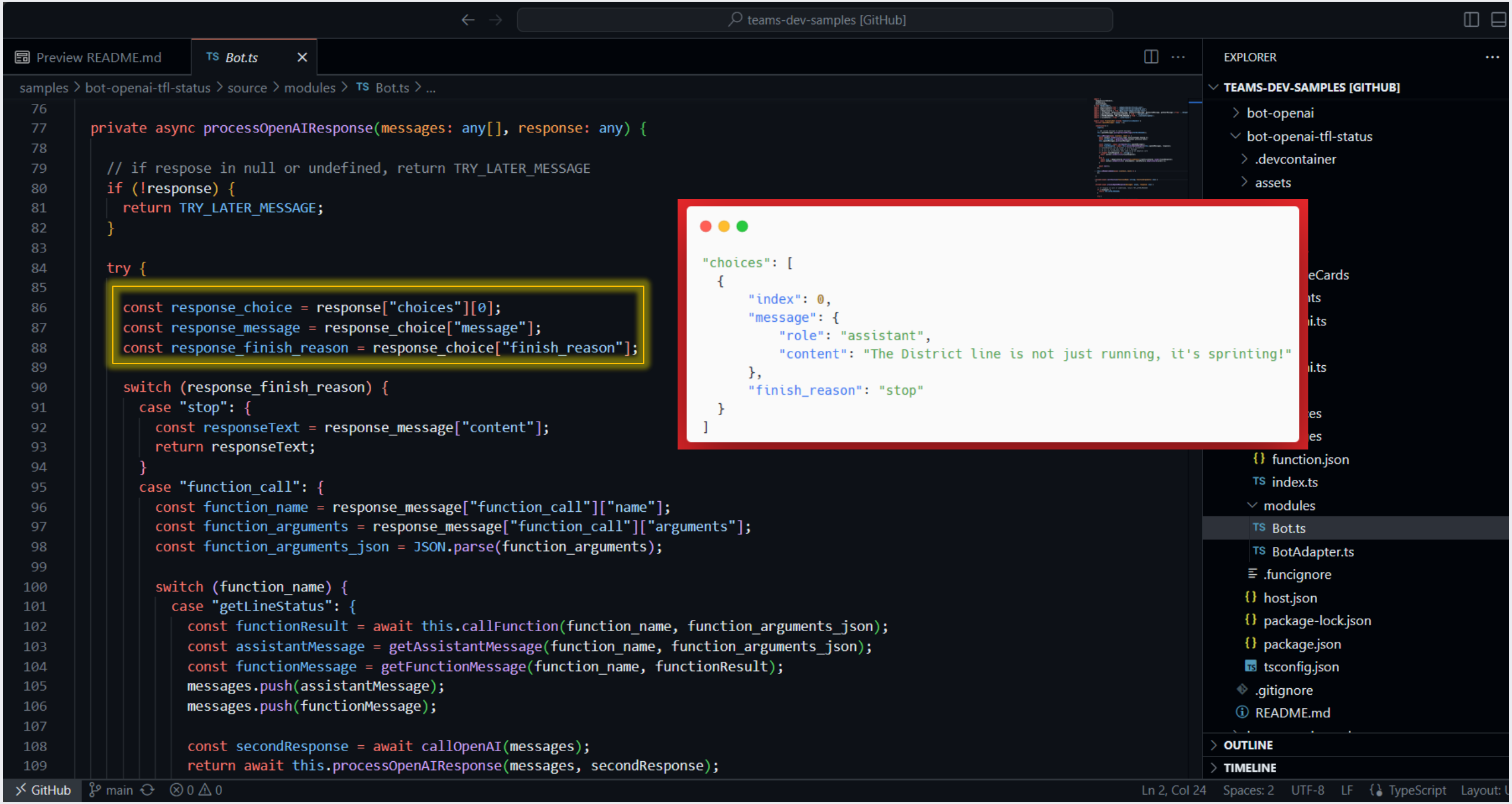

teams-dev-samples [GitHub]
Preview README.md
Bot.ts
samples > bot-openai-tfl-status > source > modules > Bot.ts > ...
private async processOpenAIResponse(messages: any[], response: any) {
// if respose in null or undefined, return TRY_LATER_MESSAGE
if (!response) {
return TRY_LATER_MESSAGE;
}
try {
const response_choice = response["choices"][0];
const response_message = response_choice["message"];
const response_finish_reason = response_choice["finish_reason"];
switch (response_finish_reason) {
case "stop": {
const responseText = response_message["content"];
return responseText;
}
case "function_call": {
const function_name = response_message["function_call"]["name"];
const function_arguments = response_message["function_call"]["arguments"];
const function_arguments_json = JSON.parse(function_arguments);
switch (function_name) {
case "getLineStatus": {
const functionResult = await this.callFunction(function_name, function_arguments_json);
const assistantMessage = getAssistantMessage(function_name, function_arguments_json);
const functionMessage = getFunctionMessage(function_name, functionResult);
messages.push(assistantMessage);
messages.push(functionMessage);
const secondResponse = await callOpenAI(messages);
return await this.processOpenAIResponse(messages, secondResponse);
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
bot-openai-tfl-status
.devcontainer
assets
manifest
source
.vscode
adaptiveCards
constants
openai.ts
helpers
openai.ts
tfl.ts
interfaces
messages
function.json
index.ts
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
Ln 2, Col 24
Spaces: 2
UTF-8
LF
TypeScript


teams-dev-samples [GitHub]
Preview README.md
Bot.ts
samples > bot-openai-tfl-status > source > modules > Bot.ts > ...
private async processOpenAIResponse(messages: any[], response: any) {
// if respose in null or undefined, return TRY_LATER_MESSAGE
if (!response) {
return TRY_LATER_MESSAGE;
}
try {
const response_choice = response["choices"][0];
const response_message = response_choice["message"];
const response_finish_reason = response_choice["finish_reason"];
switch (response_finish_reason) {
case "stop": {
const responseText = response_message["content"];
return responseText;
}
case "function_call": {
const function_name = response_message["function_call"]["name"];
const function_arguments = response_message["function_call"]["arguments"];
const function_arguments_json = JSON.parse(function_arguments);
switch (function_name) {
case "getLineStatus": {
const functionResult = await this.callFunction(function_name, function_arguments_json);
const assistantMessage = getAssistantMessage(function_name, function_arguments_json);
const functionMessage = getFunctionMessage(function_name, functionResult);
messages.push(assistantMessage);
messages.push(functionMessage);
const secondResponse = await callOpenAI(messages);
return await this.processOpenAIResponse(messages, secondResponse);
"choices": [
{
"index": 0,
"message": {
"role": "assistant",
"content": null,
"function_call": {
"name": "getTubeStatus",
"arguments": "{\n \"line\": \"district\"\n}"
}
},
"finish_reason": "function_call"
}
]
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
Ln 2, Col 24
Spaces: 2
UTF-8
LF
TypeScript


teams-dev-samples [GitHub]
Preview README.md
Bot.ts
samples > bot-openai-tfl-status > source > modules > Bot.ts > ...
private async processOpenAIResponse(messages: any[], response: any) {
// if respose in null or undefined, return TRY_LATER_MESSAGE
if (!response) {
return TRY_LATER_MESSAGE;
}
try {
const response_choice = response["choices"][0];
const response_message = response_choice["message"];
const response_finish_reason = response_choice["finish_reason"];
switch (response_finish_reason) {
case "stop": {
const responseText = response_message["content"];
return responseText;
}
case "function_call": {
const function_name = response_message["function_call"]["name"];
const function_arguments = response_message["function_call"]["arguments"];
const function_arguments_json = JSON.parse(function_arguments);
switch (function_name) {
case "getLineStatus": {
const functionResult = await this.callFunction(function_name, function_arguments_json);
const assistantMessage = getAssistantMessage(function_name, function_arguments_json);
const functionMessage = getFunctionMessage(function_name, functionResult);
messages.push(assistantMessage);
messages.push(functionMessage);
const secondResponse = await callOpenAI(messages);
return await this.processOpenAIResponse(messages, secondResponse);
"choices": [
{
"index": 0,
"message": {
"role": "assistant",
"content": null,
"function_call": {
"name": "getTubeStatus",
"arguments": "{\n \"line\": \"district\"\n}"
}
},
"finish_reason": "function_call"
}
]
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
Ln 2, Col 24
Spaces: 2
UTF-8
LF
TypeScript


teams-dev-samples [GitHub]
Preview README.md
Bot.ts
samples > bot-openai-tfl-status > source > modules > Bot.ts > ...
private async processOpenAIResponse(messages: any[], response: any) {
// if respose in null or undefined, return TRY_LATER_MESSAGE
if (!response) {
return TRY_LATER_MESSAGE;
}
try {
const response_choice = response["choices"][0];
const response_message = response_choice["message"];
const response_finish_reason = response_choice["finish_reason"];
switch (response_finish_reason) {
const getLineStatus = async (lineId: string): Promise<ILine> => {
const lineStatus = await callTfL(`https://api.tfl.gov.uk/Line/${lineId}/Status`) as ILine[];
return extractRelevantInformation(lineStatus);
};
switch (function_name) {
case "getLineStatus": {
const functionResult = await this.callFunction(function_name, function_arguments_json);
const assistantMessage = getAssistantMessage(function_name, function_arguments_json);
const functionMessage = getFunctionMessage(function_name, functionResult);
messages.push(assistantMessage);
messages.push(functionMessage);
const secondResponse = await callOpenAI(messages);
return await this.processOpenAIResponse(messages, secondResponse);
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
bot-openai-tfl-status
.devcontainer
assets
manifest
source
.vscode
adaptiveCards
constants
openai.ts
helpers
openai.ts
tfl.ts
interfaces
messages
function.json
index.ts
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
Ln 2, Col 24
Spaces: 2
UTF-8
LF
TypeScript


teams-dev-samples [GitHub]
Preview README.md
Bot.ts
samples > bot-openai-tfl-status > source > modules > Bot.ts > ...
private async processOpenAIResponse(messages: any[], response: any) {
// if respose in null or undefined, return TRY_LATER_MESSAGE
if (!response) {
return TRY_LATER_MESSAGE;
}
try {
const response_choice = response["choices"][0];
const response_message = response_choice["message"];
const response_finish_reason = response_choice["finish_reason"];
switch (response_finish_reason) {
case "stop": {
const responseText = response_message["content"];
return responseText;
}
case "function_call": {
const function_name = response_message["function_call"]["name"];
const function_arguments = response_message["function_call"]["arguments"];
const function_arguments_json = JSON.parse(function_arguments);
switch (function_name) {
case "getLineStatus": {
const functionResult = await this.callFunction(function_name, function_arguments_json);
const assistantMessage = getAssistantMessage(function_name, function_arguments_json);
const functionMessage = getFunctionMessage(function_name, functionResult);
messages.push(assistantMessage);
messages.push(functionMessage);
const secondResponse = await callOpenAI(messages);
return await this.processOpenAIResponse(messages, secondResponse);
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
bot-openai-tfl-status
.devcontainer
assets
manifest
source
.vscode
adaptiveCards
constants
openai.ts
helpers
openai.ts
tfl.ts
interfaces
messages
function.json
index.ts
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
Ln 2, Col 24
Spaces: 2
UTF-8
LF
TypeScript


teams-dev-samples [GitHub]
Preview README.md
Bot.ts
samples > bot-openai-tfl-status > source > modules > Bot.ts > ...
private async processOpenAIResponse(messages: any[], response: any) {
// if respose in null or undefined, return TRY_LATER_MESSAGE
if (!response) {
return TRY_LATER_MESSAGE;
}
try {
const response_choice = response["choices"][0];
const response_message = response_choice["message"];
const response_finish_reason = response_choice["finish_reason"];
switch (response_finish_reason) {
case "stop": {
const responseText = response_message["content"];
return responseText;
}
case "function_call": {
const function_name = response_message["function_call"]["name"];
const function_arguments = response_message["function_call"]["arguments"];
const function_arguments_json = JSON.parse(function_arguments);
switch (function_name) {
case "getLineStatus": {
const functionResult = await this.callFunction(function_name, function_arguments_json);
const assistantMessage = getAssistantMessage(function_name, function_arguments_json);
const functionMessage = getFunctionMessage(function_name, functionResult);
messages.push(assistantMessage);
messages.push(functionMessage);
const secondResponse = await callOpenAI(messages);
return await this.processOpenAIResponse(messages, secondResponse);
"choices": [
{
"index": 0,
"message": {
"role": "assistant",
"content": "The District line is not just running, it's sprinting!"
},
"finish_reason": "stop"
}
]
EXPLORER
TEAMS-DEV-SAMPLES [GITHUB]
bot-openai
bot-openai-tfl-status
.devcontainer
assets
function.json
index.ts
modules
Bot.ts
BotAdapter.ts
.funcignore
host.json
package-lock.json
package.json
tsconfig.json
.gitignore
README.md
OUTLINE
TIMELINE
GitHub
main
Ln 2, Col 24
Spaces: 2
UTF-8
LF
TypeScript

Summary

- OpenAI provides APIs. Those APIs can be used in our apps.
- Azure OpenAI provides APIs too.
- Function calling is available in OpenAI and Azure OpenAI.
- Helps us provide structured output.

Updates

- OpenAI recommend to use the “tools” object instead of “functions” in the request.
  - Postman demo
- Parallel function responses is now supported.
  - E.g. Get status of northern and central line

Resources

- OpenAI Function Calling - https://openai.com/blog/function-calling-and-other-api-updates
- Azure OpenAI Function Calling - https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/function-calling
- Sample - https://github.com/pnp/teams-dev-samples/tree/main/samples/bot-openai-tfl-status
- Sample - https://github.com/pnp/sp-dev-fx-extensions/tree/main/samples/react-application-personal-assistant