

Developer Tools

GitHub CoPilot

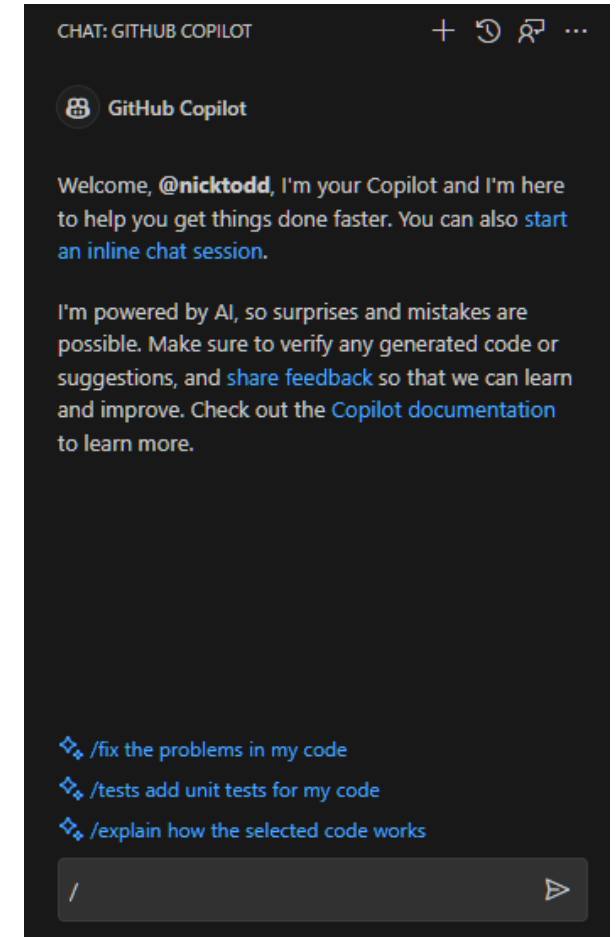
- This was the first developer tool to gain significant traction
 - <https://github.com/features/copilot>
- Monthly or Annual Fee to use
- Currently uses GPT4 LLM by default
 - Other models are available

Anthropic Claude 3.5 Sonnet in Copilot Preview

You can use the latest Claude 3.5 Sonnet model. [Learn more about how GitHub Copilot serves Claude 3.5 Sonnet.](#)

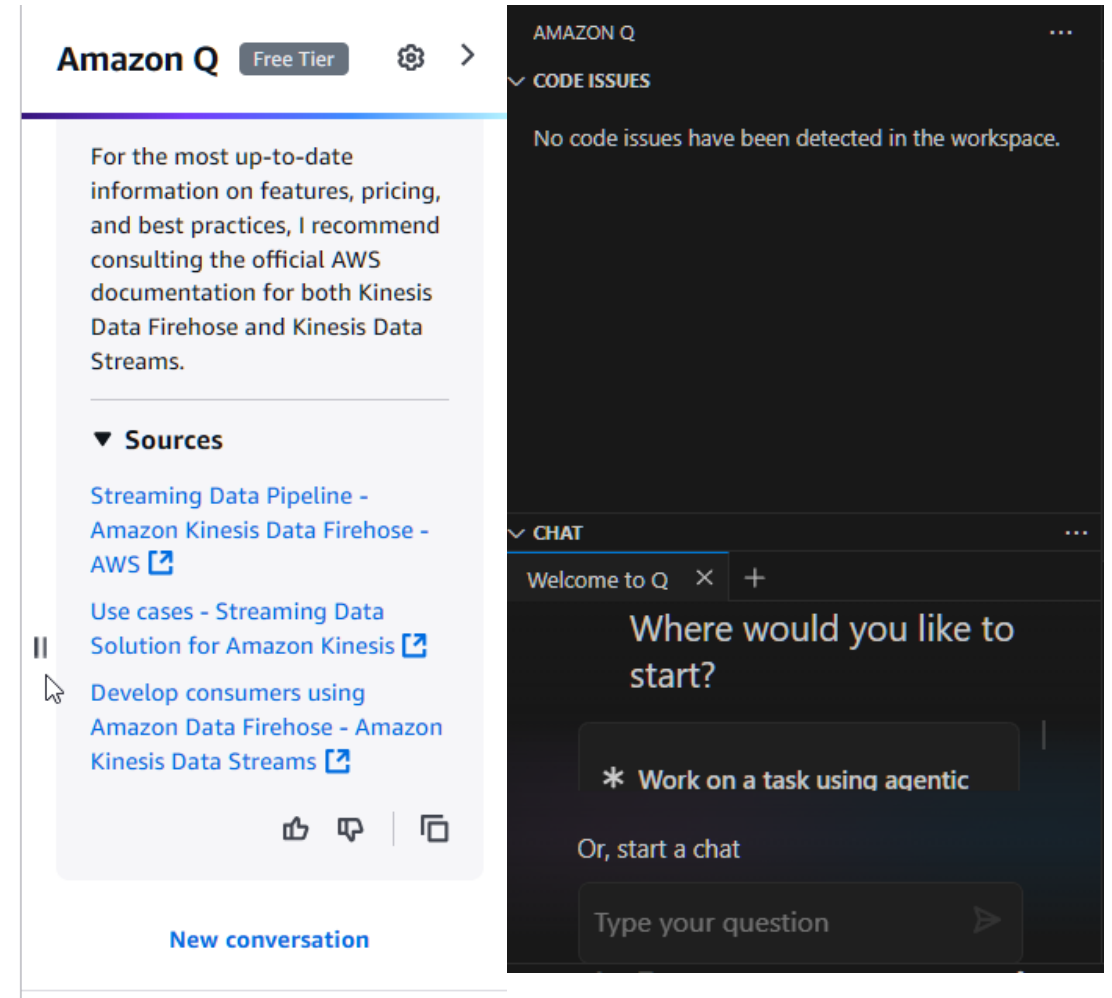
Enabled ▾

- Can be used in multiple IDEs



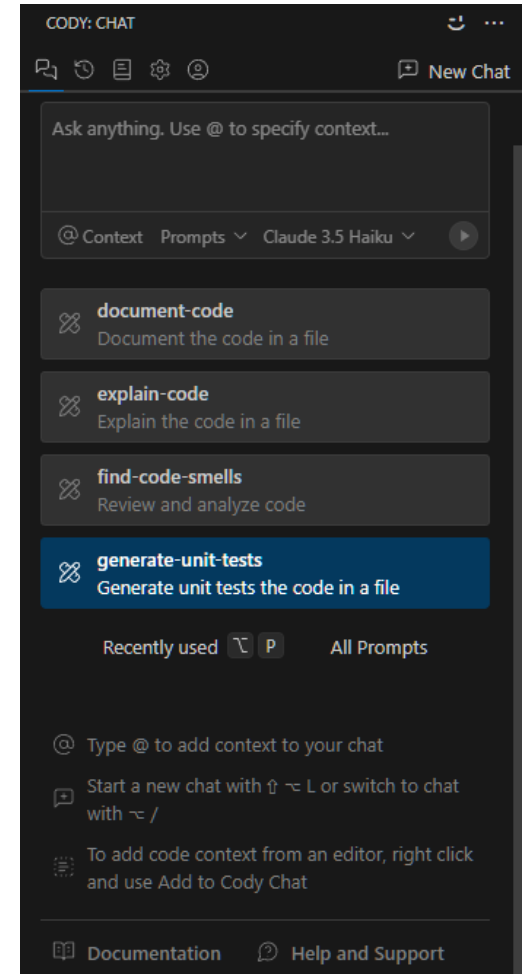
Amazon Q Developer

- The Amazon Q Developer tool provides Gen AI capabilities
 - <https://aws.amazon.com/q/developer/>
- Amazon Q Developer operates
 - In the IDE (similar to the other tools)
 - In the AWS Management Console
- You will require an AWS account in order to use it
- Multiple IDEs are supported



SourceGraph Cody

- Cody is the tool from SourceGraph
 - <https://sourcegraph.com/cody>
- Cody is different from the other offerings in that it allows the user to select an LLM for each session
- Has a free tier and then paid options

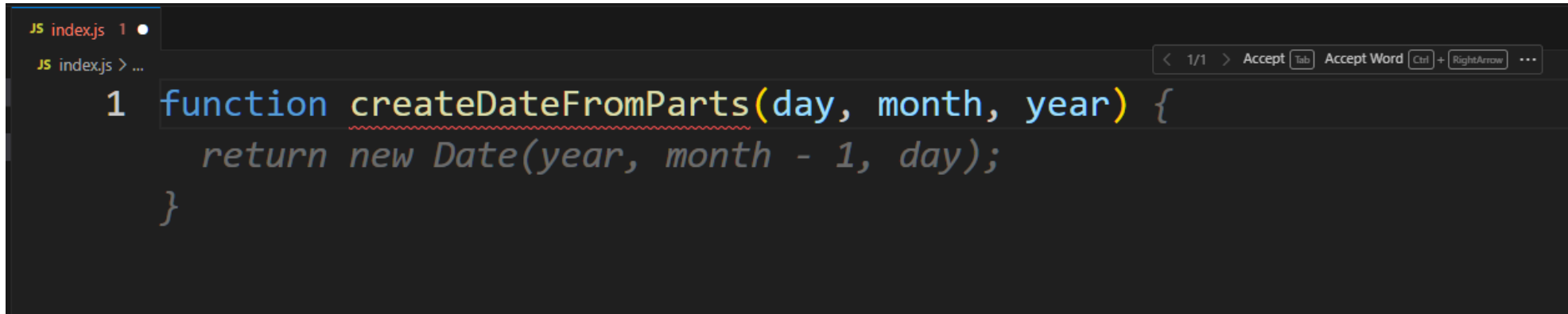


GitHub Copilot Features

- GitHub CoPilot has a growing set of features including
 - Autocompletion
 - Requesting suggestions
 - Context Menus
 - Generating documentation
 - Getting explanations
 - Fixing errors
 - Generating tests
 - Chat
 - Project creation

AutoCompletion

- One of the most useful features of CoPilot is the autocompletion capability
 - You begin typing some code and it suggests completions



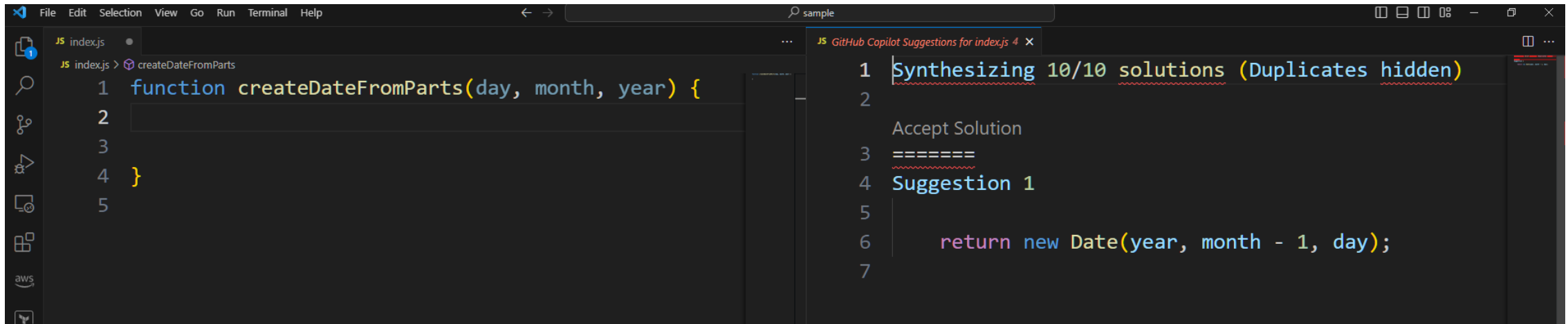
The screenshot shows a code editor with a dark theme. The file name 'index.js' is visible in the top left. The code being typed is a JavaScript function: `function createDateFromParts(day, month, year) {`. The word `createDateFromParts` is highlighted in yellow, and a red squiggly line is underneath it, indicating a suggestion. The function body contains `return new Date(year, month - 1, day);`. The closing brace `}` is on the next line. In the top right corner, there is a toolbar with buttons for navigation and actions: '< 1/1 > Accept Tab Accept Word Ctrl + RightArrow ...'.

```
JS index.js 1 •
JS index.js > ...
1 function createDateFromParts(day, month, year) {
    return new Date(year, month - 1, day);
}
```

- You can then **Accept** the suggestion using **Tab**

Request Suggestions

- Using **Ctrl-Enter**, you can request suggestions from GitHub CoPilot



The screenshot shows the Visual Studio Code editor with a file named `index.js`. The code in the editor is:

```
1 function createDateFromParts(day, month, year) {  
2  
3  
4 }  
5
```

On the right side, a panel titled "GitHub Copilot Suggestions for index.js 4" displays the following suggestions:

```
1 Synthesizing 10/10 solutions (Duplicates hidden)  
2  
3 Accept Solution  
4 =====  
5 Suggestion 1  
6     return new Date(year, month - 1, day);  
7
```

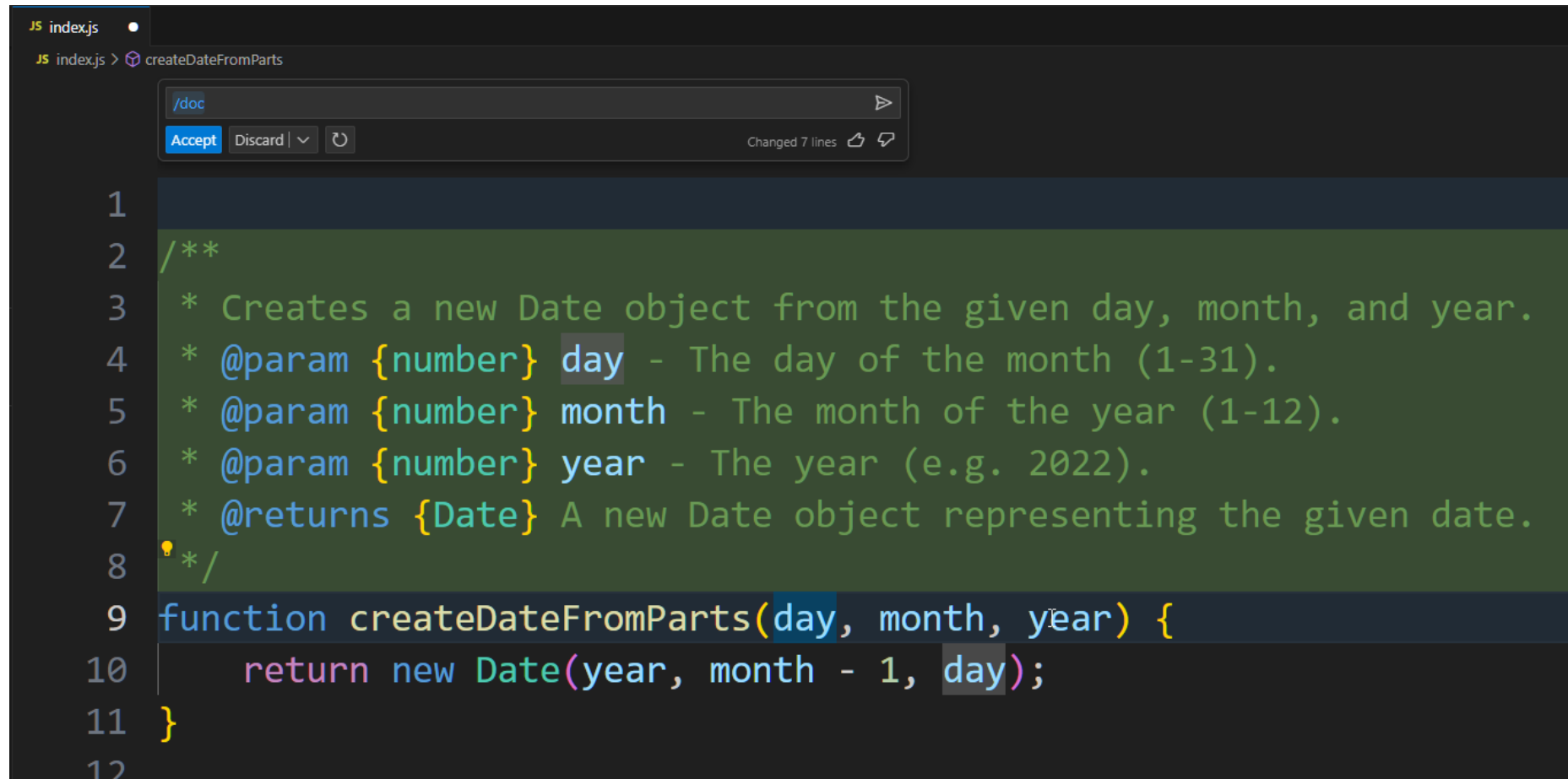
- Multiple suggestions are then displayed and you can select the preferred suggestion

Context Menu

- You can bring up the GitHub CoPilot context menu using **Ctrl-I**
- From this menu you can then enter one of the following commands
 - `/doc`
 - `/explain`
 - `/fix`
 - `/test`

Using /doc to Generate Documentation

- Using `/doc` you can generate documentation for your code

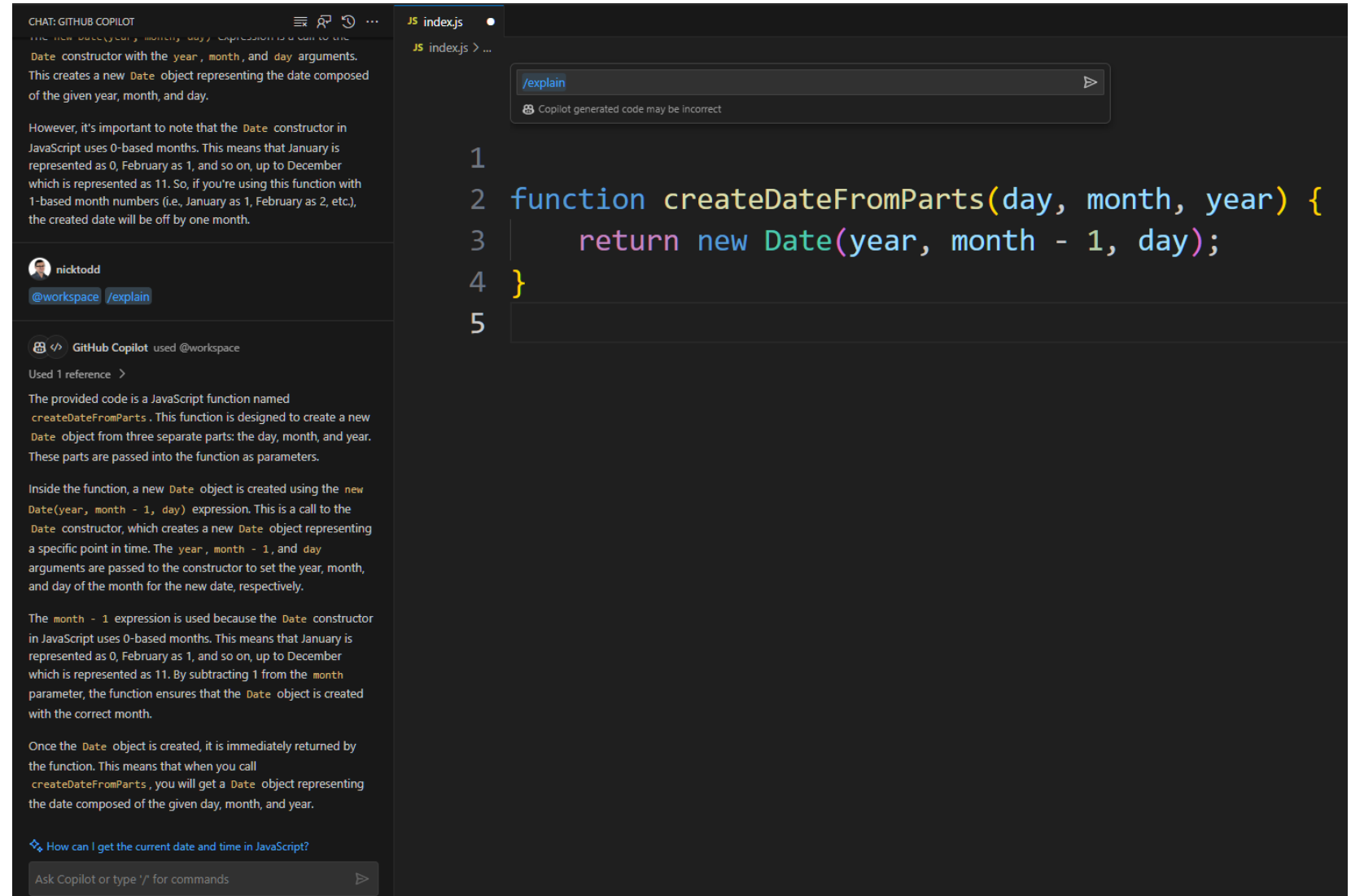


The screenshot shows a code editor with a file named `index.js`. A command palette is open, showing the `/doc` command selected. Below the command palette, the code for the `createDateFromParts` function is visible. The function is defined as follows:

```
1  
2 /**  
3  * Creates a new Date object from the given day, month, and year.  
4  * @param {number} day - The day of the month (1-31).  
5  * @param {number} month - The month of the year (1-12).  
6  * @param {number} year - The year (e.g. 2022).  
7  * @returns {Date} A new Date object representing the given date.  
8  */  
9 function createDateFromParts(day, month, year) {  
10     return new Date(year, month - 1, day);  
11 }  
12
```

Using /explain to Understand

- Using `/explain` describes the following code in the GitHub CoPilot Chat window



Fixing Problems

- Let's introduce a deliberate problem with our function

```
function createDateFromParts(day, month, year)
{
    return new Date(year, month-1, day);
}
```

```
function createDateFromParts(day, month, year)
{
    return new Date(year, month, day);
}
```

Using CoPilot to Fix the Issue

- The code now has the error
- I can ask GitHub CoPilot to fix the error with `/fix` followed by a description of the problem

The image shows a VS Code editor with a JavaScript file named `index.js`. The initial code is as follows:

```
1  
2 function createDateFromParts(day, month, year) {  
3   return new Date(year, month, day);  
4 }
```

A red box highlights the `month` parameter in the `Date` constructor. A Copilot chat window is open with the prompt `/fix the months are coming up wrong`. Below the prompt, it says "Copilot generated code may be incorrect".

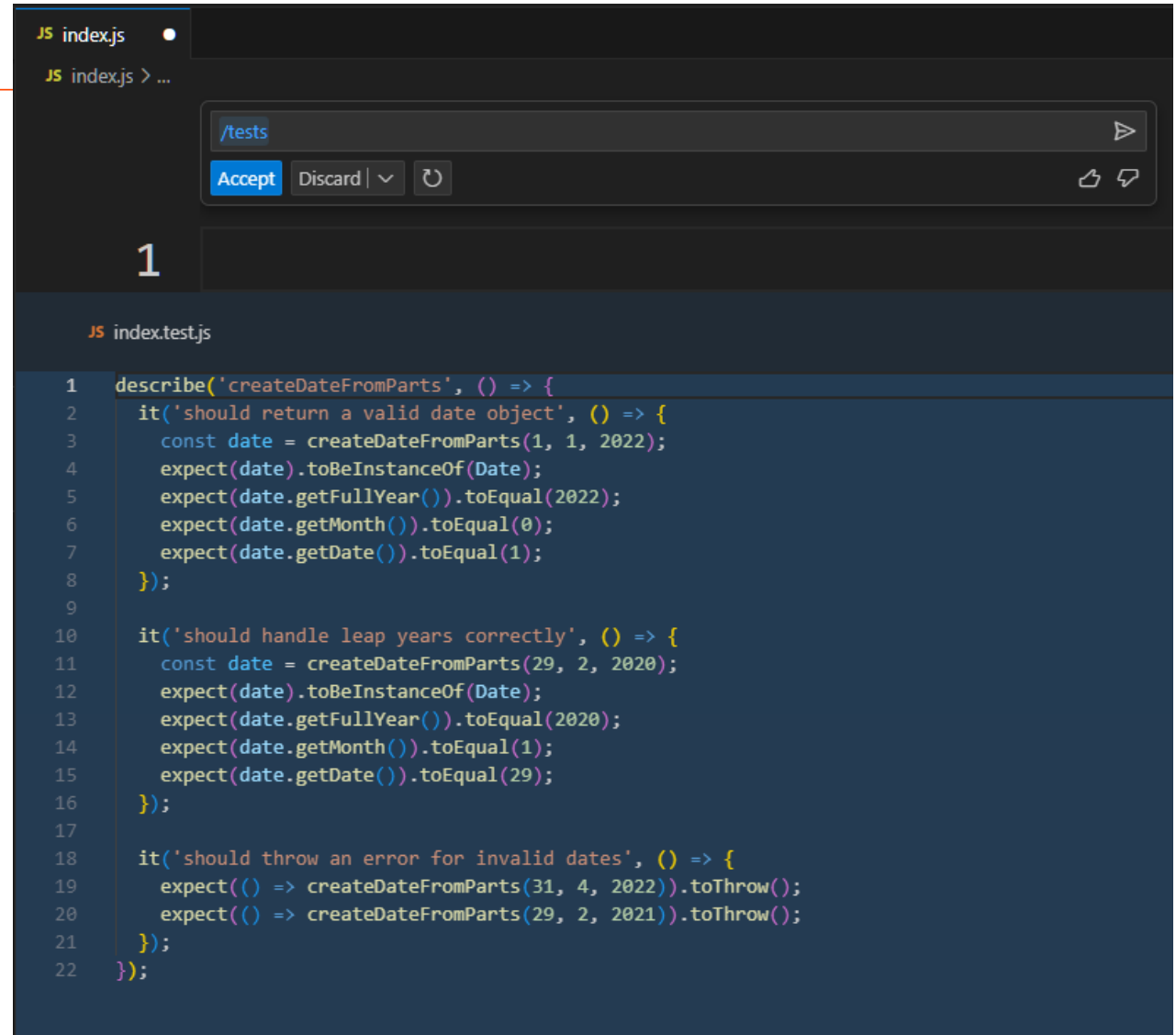
The second part of the image shows the result after clicking the `Accept` button. The code is now:

```
1  
2 function createDateFromParts(day, month, year) {  
3-   return new Date → 3+   return new Date(year, month - 1, day);  
5 }
```

A red box highlights the change from `month` to `month - 1`. The Copilot chat window shows the `Accept` button, a `Discard` button, and a refresh icon, along with the text "Changed 1 line".

Creating Tests

- You can generate tests using `/tests`

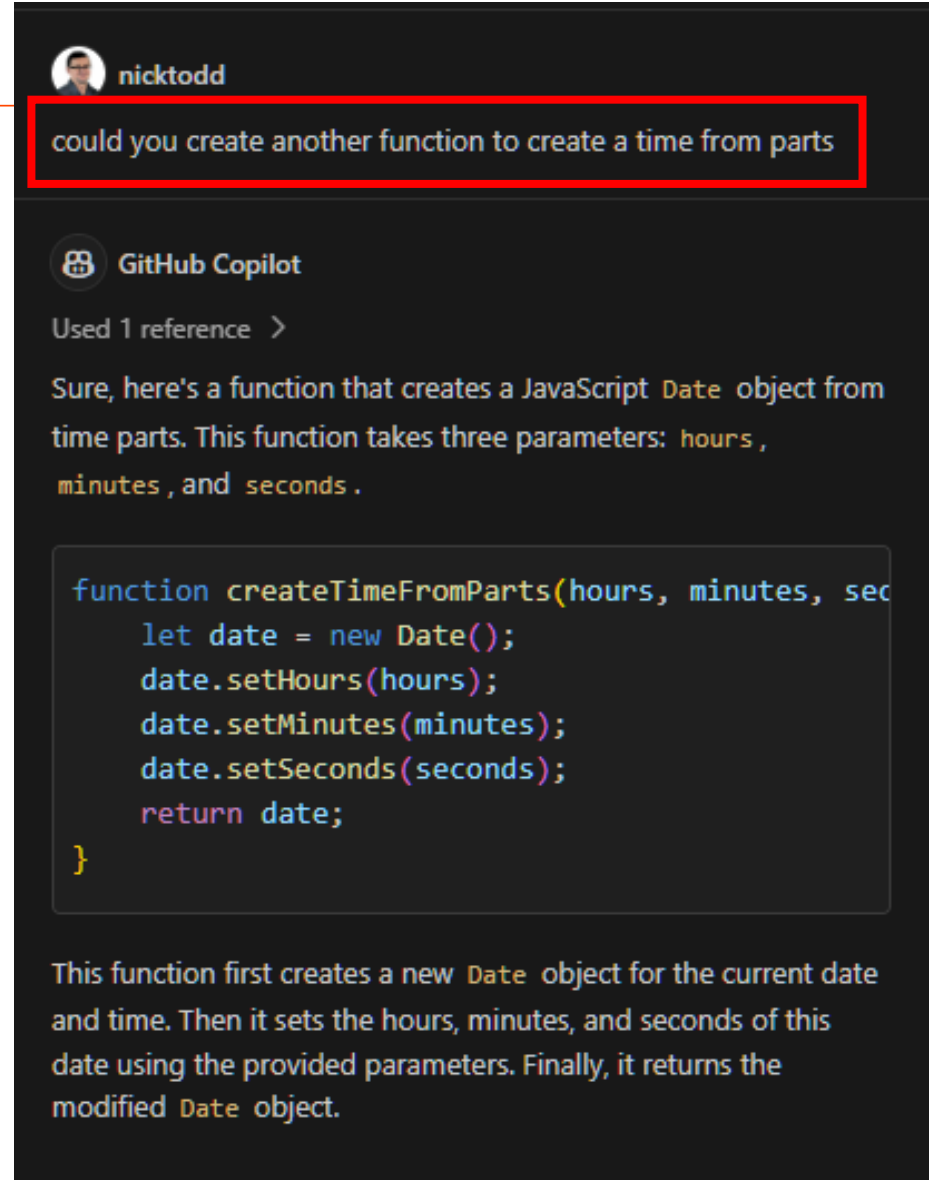


The screenshot shows the VS Code interface. At the top, a terminal window displays the command `/tests` being executed, with buttons for `Accept`, `Discard`, and a refresh icon. Below the terminal, a file explorer shows a new file named `index.test.js` created. The main editor area displays the content of `index.test.js`, which contains three Jest test cases for a `createDateFromParts` function.

```
1 describe('createDateFromParts', () => {
2   it('should return a valid date object', () => {
3     const date = createDateFromParts(1, 1, 2022);
4     expect(date).toBeInstanceOf(Date);
5     expect(date.getFullYear()).toEqual(2022);
6     expect(date.getMonth()).toEqual(0);
7     expect(date.getDate()).toEqual(1);
8   });
9
10  it('should handle leap years correctly', () => {
11    const date = createDateFromParts(29, 2, 2020);
12    expect(date).toBeInstanceOf(Date);
13    expect(date.getFullYear()).toEqual(2020);
14    expect(date.getMonth()).toEqual(1);
15    expect(date.getDate()).toEqual(29);
16  });
17
18  it('should throw an error for invalid dates', () => {
19    expect(() => createDateFromParts(31, 4, 2022)).toThrow();
20    expect(() => createDateFromParts(29, 2, 2021)).toThrow();
21  });
22 });
```

GitHub CoPilot Chat

- In addition to the /tests /fix /doc /explain options, there is also a Chat option
- The Chat option is like using ChatGPT capability but context sensitive and within the IDE



The screenshot shows a chat interface with a dark background. At the top, a user profile for 'nicktodd' is visible. Below it, a red-bordered box contains the user's message: 'could you create another function to create a time from parts'. The response from 'GitHub Copilot' follows, indicating it used 1 reference. The response text explains that it will create a JavaScript Date object from time parts, taking hours, minutes, and seconds as parameters. Below the text is a code block containing a JavaScript function named 'createTimeFromParts'. The function takes three arguments: 'hours', 'minutes', and 'seconds'. It creates a new Date object, sets its hours, minutes, and seconds using the provided parameters, and returns the modified Date object. A final paragraph explains the function's logic: it creates a new Date object for the current date and time, then sets the hours, minutes, and seconds using the provided parameters, and finally returns the modified Date object.

nicktodd

could you create another function to create a time from parts

GitHub Copilot

Used 1 reference >

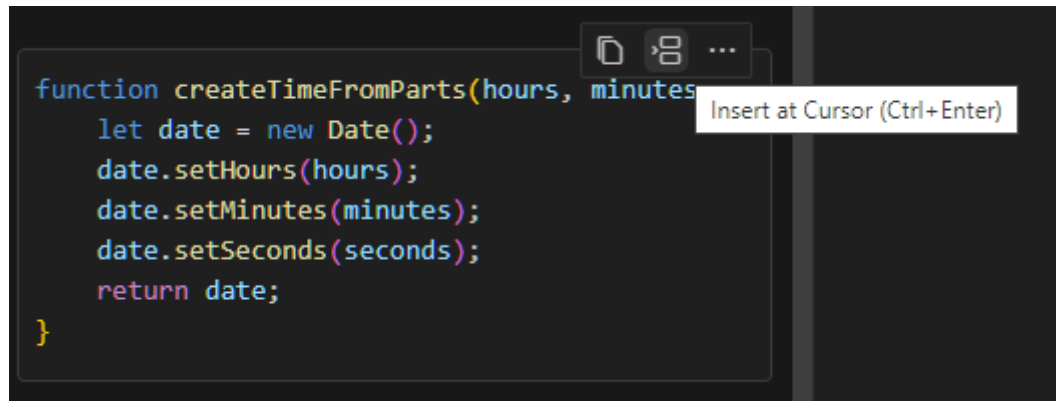
Sure, here's a function that creates a JavaScript `Date` object from time parts. This function takes three parameters: `hours`, `minutes`, and `seconds`.

```
function createTimeFromParts(hours, minutes, seconds) {
  let date = new Date();
  date.setHours(hours);
  date.setMinutes(minutes);
  date.setSeconds(seconds);
  return date;
}
```

This function first creates a new `Date` object for the current date and time. Then it sets the hours, minutes, and seconds of this date using the provided parameters. Finally, it returns the modified `Date` object.

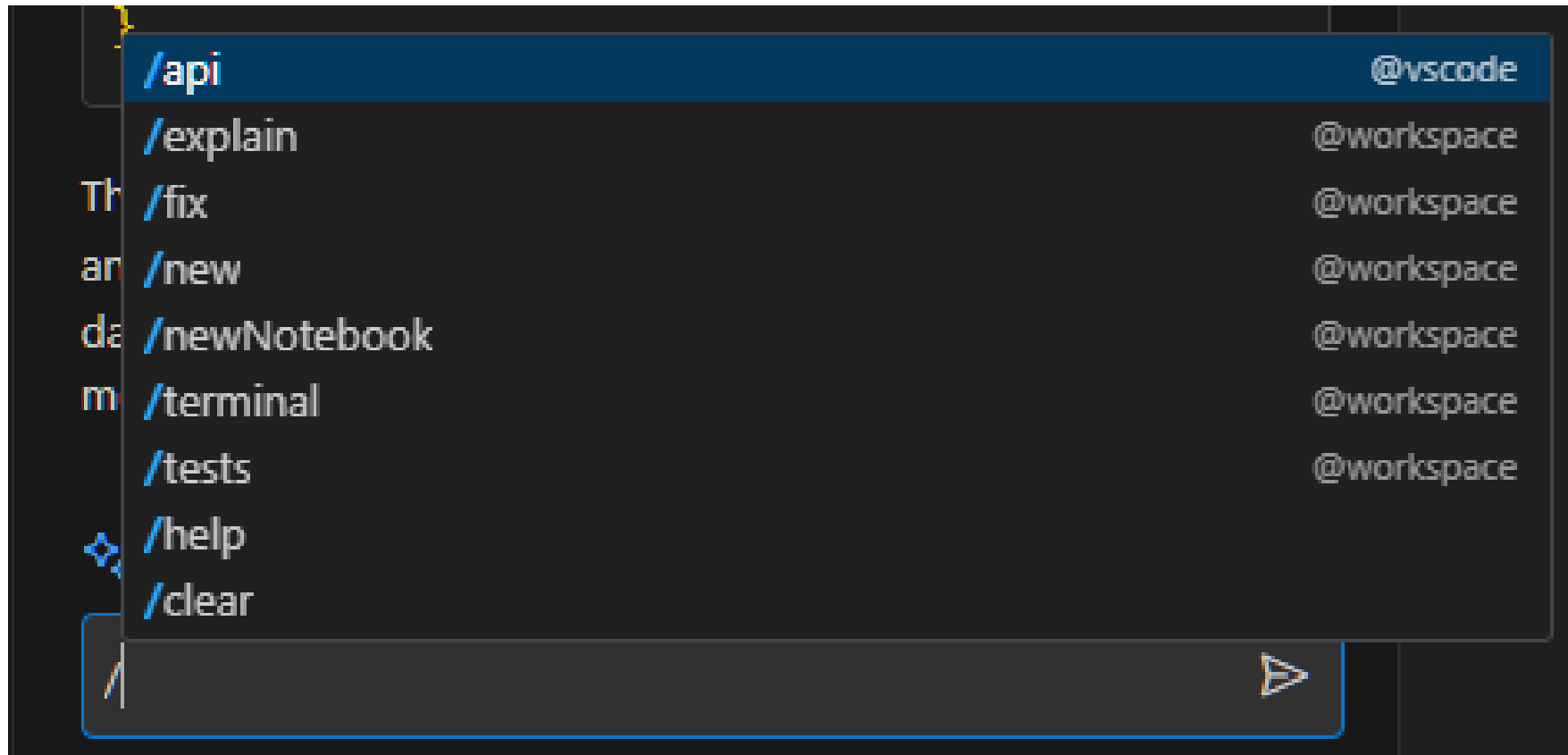
Using the Suggestions

- The code can then be inserted into your codebase either at the cursor, or alternatively just copied to the clipboard



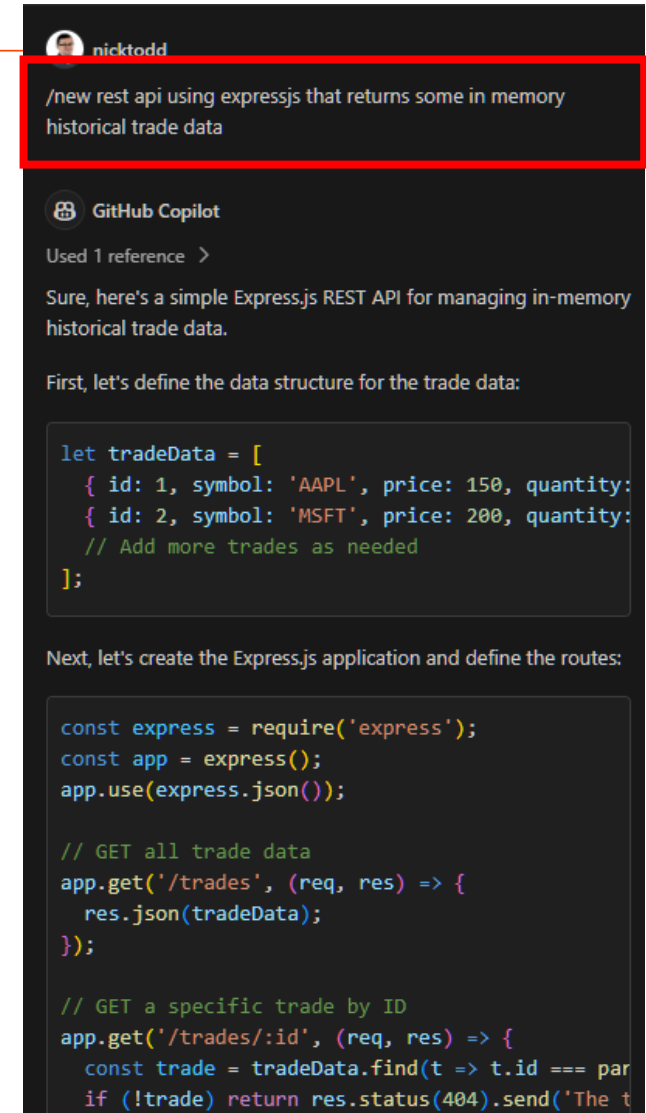
Chat Commands

- The Chat also introduces a number of additional commands



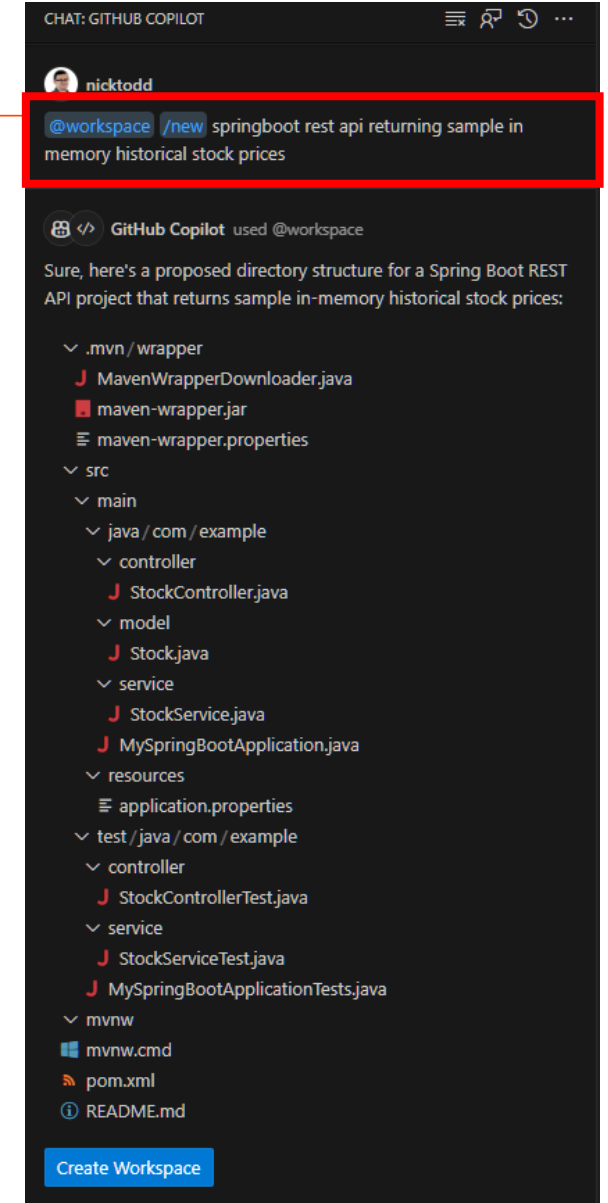
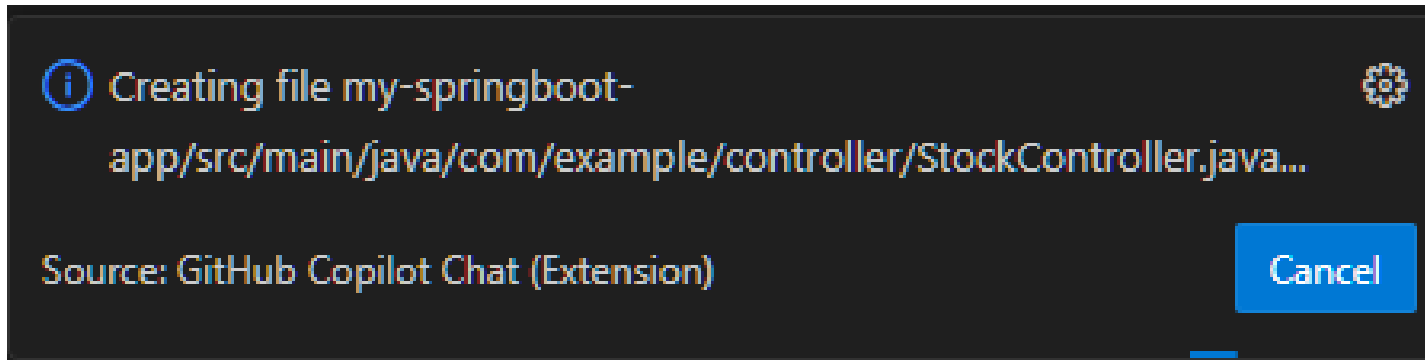
Creating Projects using /new

- The `/new` option can be used to create a project from scratch
 - If you type `/new` you will get suggested code fragments



Creating the Entire Project

- Entire project scaffolds can be created as well if you use the **@workspace** keyword before the **/new** command
- For example, you can create an entire SpringBoot project



Errors in the Code

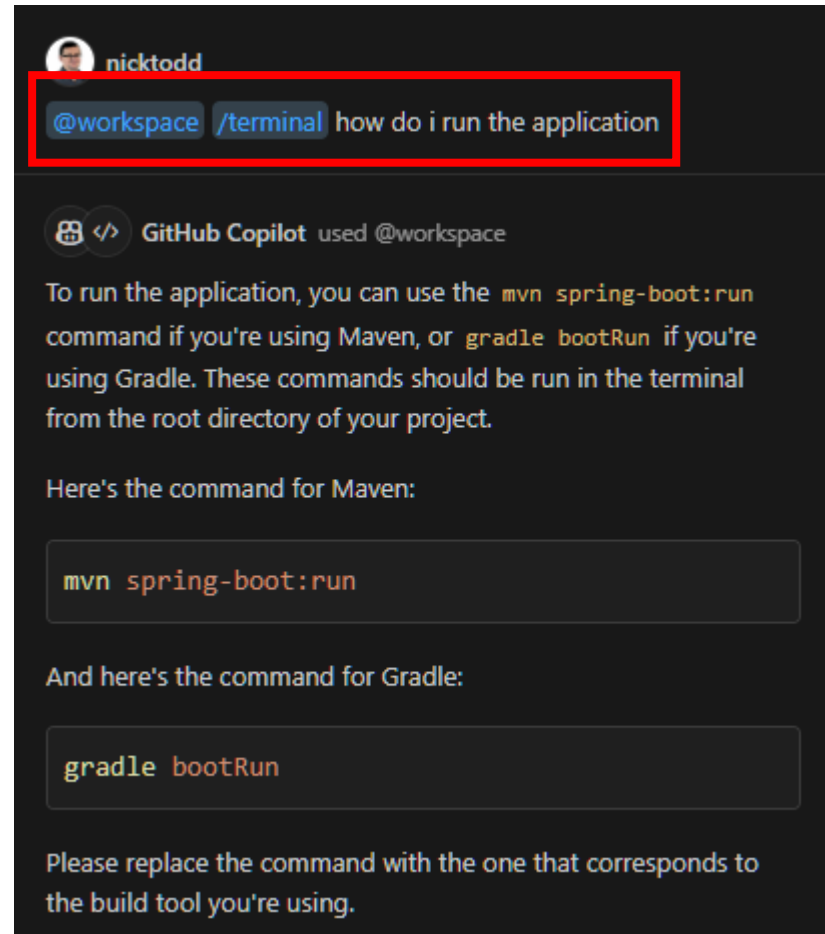
- Although it generates the required scaffolding, it won't necessarily be perfect!
- In the example generated, the entity class had different properties to those used by the constructor call in the service layer

```
@Service
public class StockService {

    private List<Stock> stocks = Arrays.asList(
        new Stock(1L, "Apple Inc.", 150.0),
        new Stock(2L, "Microsoft Corporation", 280.0),
        new Stock(3L, "Amazon.com, Inc.", 3300.0)
    );
}
```

GitHub CoPilot and the Terminal

- Using `@workspace` /`/terminal` you can ask questions about what to enter at the terminal
- At time of writing, GitHub are creating a CLI AI as well which will mean that you can request AI assistance from the terminal directly



Emphasising the Co in CoPilot

How it Works

How It Works

- GitHub Copilot offers suggestions from a model that OpenAI built from billions of lines of open source code
- That means your code will have to be sent to the model in order to get suggestions
- If using it in your own account, you can additionally control whether your code is used to improve the model

☒ Allow GitHub to use my code snippets for product improvements *

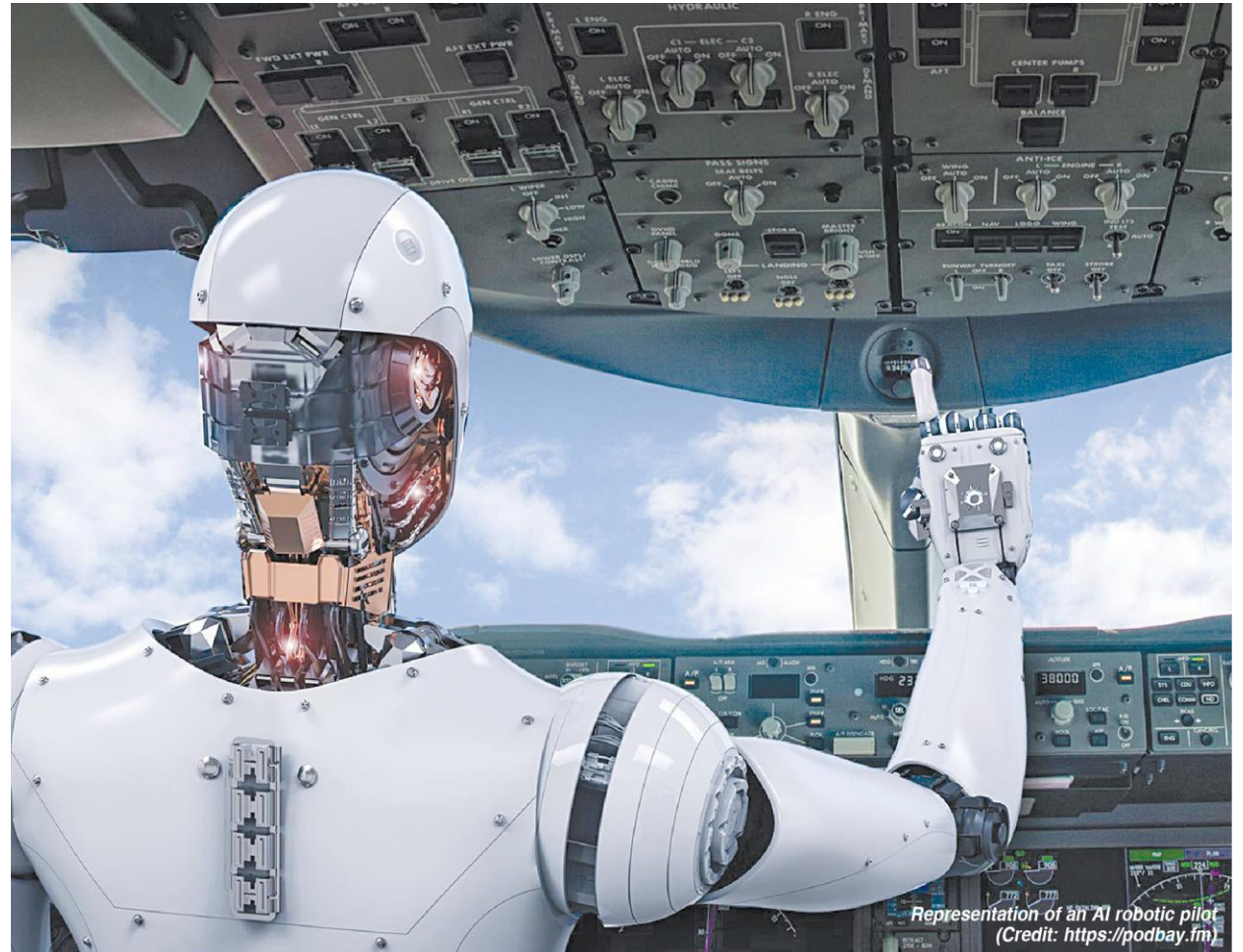
Allow GitHub, its affiliates and third parties to use my code snippets to research and improve GitHub Copilot suggestions, related models and product features.

More information in [Privacy FAQ](#).

Save

Pilot Flying / Pilot Monitoring

- In commercial aviation there is always
 - A pilot flying
 - A pilot monitoring
- If you are using GitHub CoPilot who is the pilot monitoring?
 - YOU!!
- You CANNOT abdicate responsibility for your application logic to the AI
 - It's not good enough *yet!*



Representation of an AI robotic pilot
(Credit: <https://podbay.fm>)

Precautions

- Code Quality
 - The training set for GitHub Copilot may contain insecure coding patterns, bugs, or references to outdated APIs or idioms
 - When GitHub Copilot produces suggestions based on this training data, those suggestions may also contain undesirable patterns
- Responsibilities
 - You are responsible for ensuring the security and quality of your code
 - GitHub recommend the following precautions are followed
 - Rigorous testing
 - IP scanning
 - Tracking for security vulnerabilities


Best Practices

- Effective Prompts
 - Make it clear
 - Break it down
 - Provide examples
 - Consider Neighbouring tabs
 - Consider code above and below – Fill in the Middle
 - Good naming
- Testing
- Regular Expressions


Prompting - Make it Clear

- Give good context prompts
 - *I want a function that returns a date from parts*
 - *I want a function that returns a date from a day month and year passed as parameters*

```
JS GitHub Copilot Suggestions for index.js 3 x
1 Synthesizing 9/10 solutions
2
3 Accept Solution
4 =====
5 Suggestion 1
6 // I want a function that returns a date from parts.
7 //
8 //
9
10 Accept Solution
11 =====
12 Suggestion 2
13 // I want a function that returns a date from parts
14 // I want a function that returns a date from a string
15 // I want a function that returns a date from a date
16
17 const date = require('./date');
18
19 const dateFromParts = date.dateFromParts;
20 const dateFromString = date.dateFromString;
21 const dateFromDate = date.dateFromDate;
22
23 module.exports = {
24   dateFromParts,
25   dateFromString,
26   dateFromDate,
27 };
28
```



```
Synthesizing 9/10 solutions
Accept Solution
=====
Suggestion 1
// I want a function that returns a date from a day month and year passed as parameters.
// This is how I would do it:
function myFunction(day, month, year) {
  return new Date(year, month - 1, day);
}
console.log(myFunction(1, 2, 2020));
// If you run this code you will see that it returns the date 2020-01-31.
```



Prompting - Break it Down

- Break your request down into bytesize chunks

```
// I want a class for a Car  
// with a make and model and get and set methods for the properties  
// i also want a speed property that defaults to 0  
// and I want an accelerate method that increments the speed by 10  
// i also want a brake method that decrements the speed by 10
```



Prompting - Provide Examples

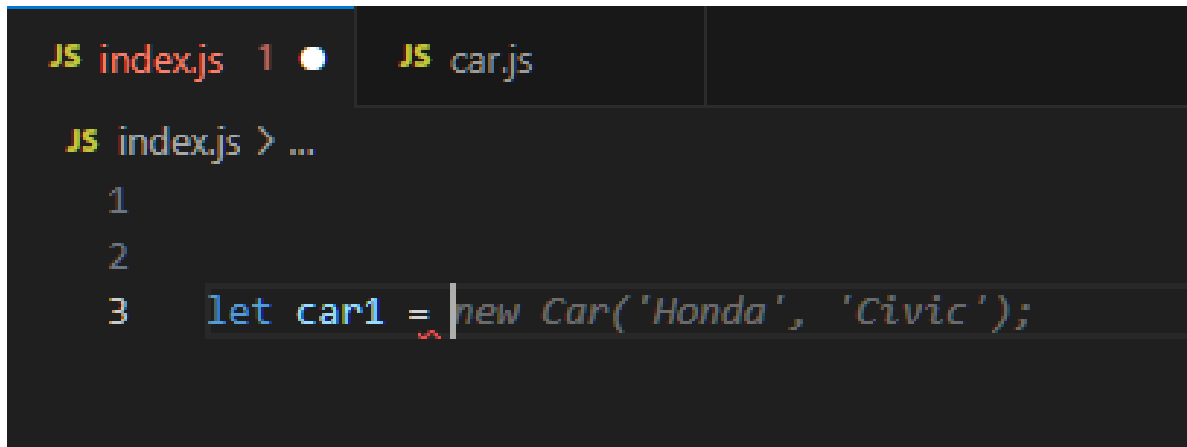
- When prompting, provide examples
 - Copilot will also help you with your prompts!!

```
let people = [  
  { name: 'John', age: 21 },  
  { name: 'Peter', age: 31 },  
  { name: 'Sarah', age: 41 },  
]  
  
// i want an array of just the names eg. ['John', 'Peter', 'Sarah']
```



Prompting - Neighbouring Tabs

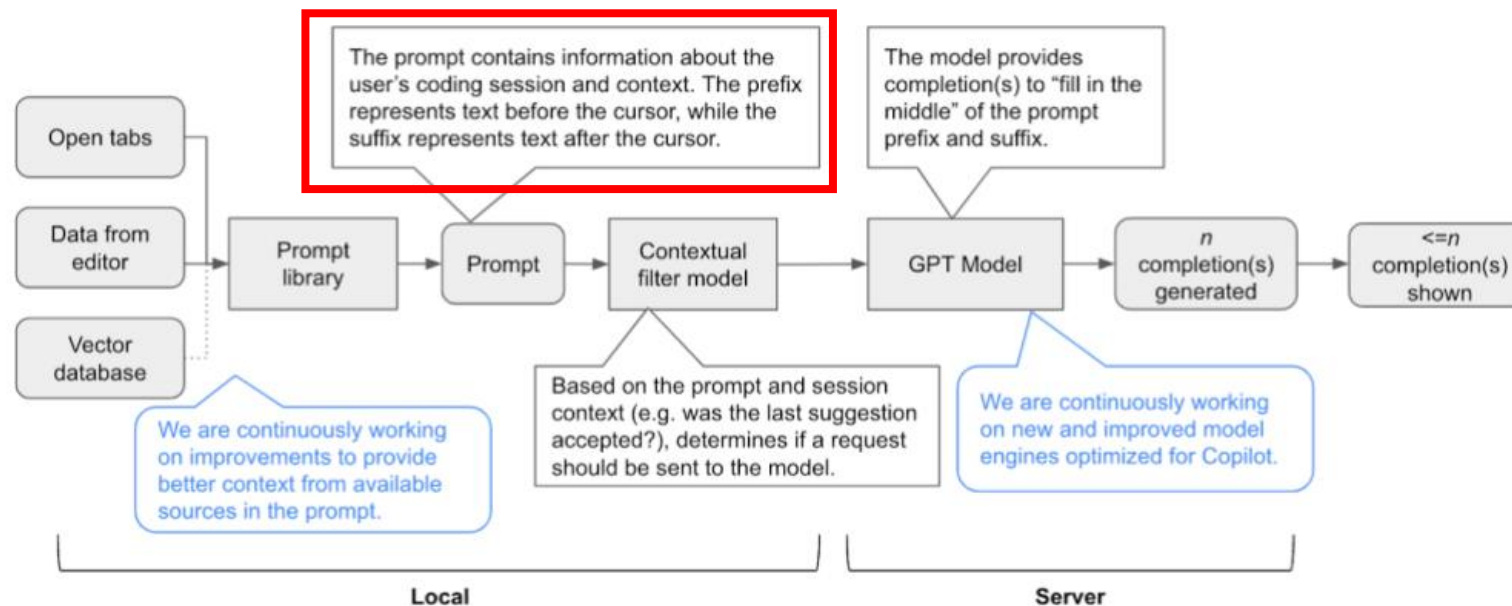
- CoPilot has a feature called Neighbouring tabs
 - *“Neighboring tabs is what we call the technique that allows GitHub Copilot to process all of the files open in a developer’s IDE instead of just the single one the developer is working on. By opening all files relevant to their project, developers automatically invoke GitHub Copilot to comb through all of the data and find matching pieces of code between their open files and the code around their cursor—and add those matches to the prompt”*



```
JS index.js 1 JS car.js
JS index.js > ...
1
2
3 let car1 = new Car('Honda', 'Civic');
```

Prompting - Fill in the Middle


- The Code that is sent to GitHub includes the code before *and after* your cursor so it gets better context
 - This is referred to as the **Fill in the Middle Paradigm**




Simplified system diagram focused on model quality efforts. Made by Alice Li, machine learning researcher at GitHub.

Prompting - Use Good Naming


- As we all know from our development experience, good naming is important to understand code
 - This is also true for your CoPilot!



```
function createDateFromParts(year, month, day) {  
    return new Date(year, month - 1, day)  
}
```



```
function makeDate() {  
    var d = new Date();  
    var formattedDate = "";  
  
    formattedDate += (d.getMonth() + 1) + "_";  
    formattedDate += d.getDate() + "_";  
    formattedDate += d.getFullYear();  
  
    return formattedDate;  
}
```



Testing

- Generating tests - be careful
- Consider the simple date from parts example, this could be what you do
 1. Generate some code to create the date (it uses the month with index 0)
 2. You then generate a test (it will simply test that the month parses with an index of 0)
 3. The test will pass
- The code is however wrong - you trusted your copilot

1. Create an Incorrect Function

- This should be **month -1** but you didn't notice

```
// export the createDateFromParts function
module.exports = createDateFromParts;

// function to create a date from the parts
function createDateFromParts(day, month, year)
{
    return new Date(year, month, day);
}
```



2. Now Create the Incorrect Test – That Passes!

- It will create a passing test for your incorrect function
 - The test uses 0s for the months because the function being tested does!
- Everything passes!!


Suggestion 2

```
// create a unit test for the index.js file
//
// Path: index.test.js
//
// // import the function
// const createDateFromParts = require('./index.js');
//
// // test the function
// describe('createDateFromParts()', () => {
//   it('creates a valid date', () => {
//     expect(createDateFromParts(1, 0,
// 2000)).toEqual(new Date(2000, 0, 1));
//   });
// });
```




Testing - An Alternative Approach

1. When generating tests, consider describing the test that is required without referring to your code being tested



```
// create a unit test for a function called createDateFromParts
// the function is in a file called index.js
// the parts are day, month and year
// and the month should account for the offset of 1
```

Suggestion 1



```
const createDateFromParts = require('./index');

test('should return a date object', () => {
  expect(createDateFromParts(1, 1, 2020)).toBeInstanceOf(Date);
});

test('should return a date object for 1st Jan 2020', () => {
  expect(createDateFromParts(1, 1, 2020)).toEqual(new Date(2020, 0, 1));
});
```

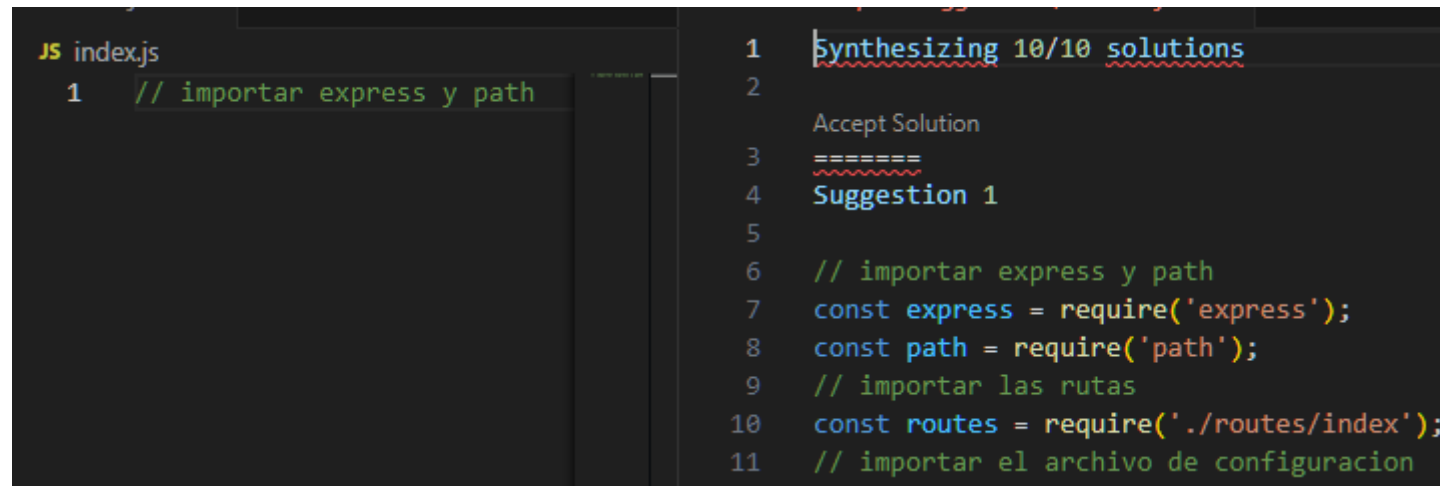
Regular Expressions

- GitHub CoPilot is very good at creating regular expressions and can save you a lot of time

```
JS index.js > ...  
1  function validatePhoneNumber (phoneNumber) {  
    const regExp = /^\\d{10}$/;  
    return regExp.test(phoneNumber);  
2  
3  }
```

Translation

- For non-English speakers, you can use CoPilot in different languages



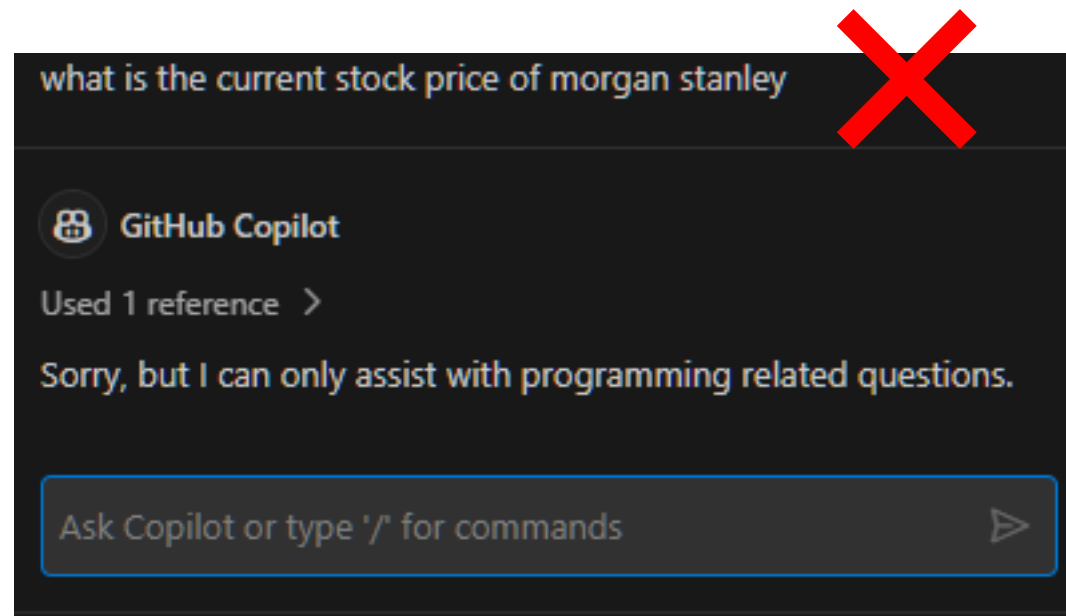
The screenshot shows a code editor with two panels. The left panel shows a file named `index.js` with the first line `// importar express y path`. The right panel shows a suggestion for the same line, labeled `Synthesizing 10/10 solutions` and `Suggestion 1`. The suggested code is in Spanish and uses `require` for imports.

```
JS index.js
1 // importar express y path

1 Synthesizing 10/10 solutions
2
3 Accept Solution
4 =====
5 Suggestion 1
6 // importar express y path
7 const express = require('express');
8 const path = require('path');
9 // importar las rutas
10 const routes = require('./routes/index');
11 // importar el archivo de configuracion
```

Don't use it as a Chat GPT Alternative!

- Do not try and use it for questions outside of the coding space
- It is not designed for that



Summary

- Overview of Gen AI Developer Tools
- Features of Github CoPilot
- Understanding the importance of the Co in CoPilot
- Best Practices