



SOAR TO NEW HEIGHT DEVS! MASTER CODING WITH GITHUB COPILOT AND CHAT



Randy Pagels

DevOps Architect | Trainer
Randy.Pagels@Xebia.com



Copilot

AI Development Assistant
copilot@github.com

Xebia | Microsoft
Services

Thank you to our Sponsors!





GITHUB
COPILOT

1

GITHUB COPILOT AI PAIR PROGRAMMER



Visual Studio



Visual Studio
Code



Vim/Neovim



JetBrains



Azure Data
Studio

2

GITHUB COPILOT LICENSES



Individual

\$10

per user / month



Business

\$19

per user / month



Enterprise

\$39

per user / month

* Requires GitHub Enterprise Cloud.

** Currently in preview. (available in 2024)

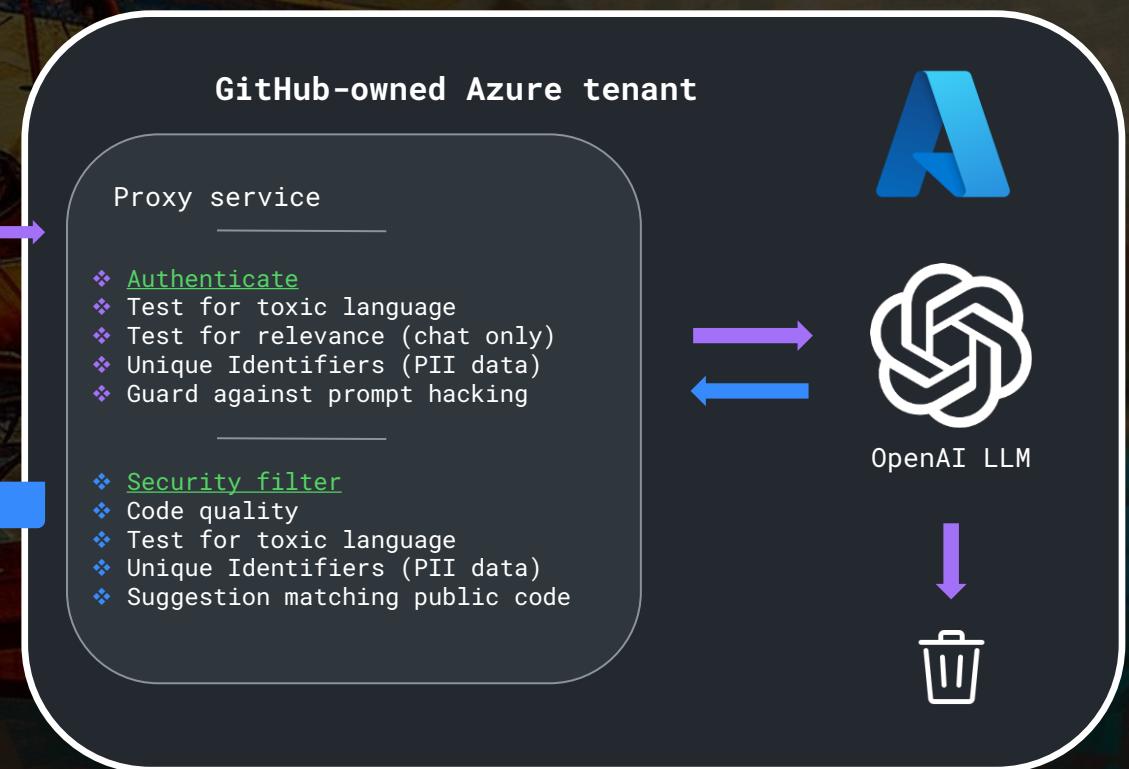
3

GITHUB COPILOT HOW DOES IT WORK

The screenshot shows a GitHub Copilot interface. At the top, there are four tabs: `runtime.go`, `course.rb`, `time.js`, and `IsPrimeTest.java`. Below the tabs is a code editor with the following Go code:

```
1 package main
2
3 type Run struct {
4     Time int // in milliseconds
5     Results string
6     Failed bool
7 }
8
9 // Get average runtime of successful runs in seconds
10 func averageRuntimeInSeconds(runs []Run) float64 {
11
12
13
14
15
16
17
18
19
20
21
22
23 }
```

A blue arrow points from the code editor to a purple box labeled "Context". Another blue arrow points from the "Context" box to a blue box labeled "Suggestions". A "Copilot" button is visible at the bottom left.



4

GITHUB COPILOT GPT

Generative Pre-trained Transformer (type of LLM)

Developed by OpenAI

GPT-3.5

- GitHub Copilot Completions
- Fast
- Cheap(er)
- Max Tokens (16k)
- Trained on Sep 2021

GPT-4

- GitHub Copilot Chat
- 10X Powerful
- Accurate
- Max Tokens (32k)
- Trained on Dec 2023

The background of the image is a dark, sepia-toned photograph of several vintage biplane aircraft parked on a tarmac. The planes are silhouetted against a lighter sky, with some wispy clouds visible. The overall mood is mysterious and historical.

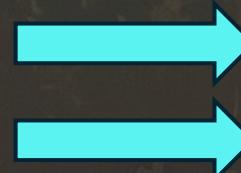
LLM

1

WHAT IS AN LLM? LARGE LANGUAGE MODEL

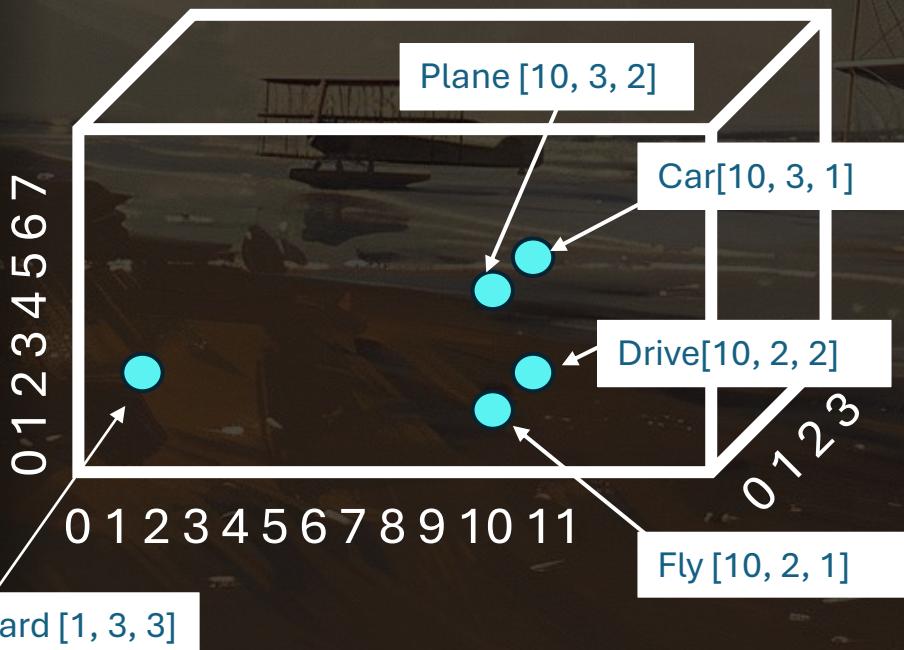
"A plane flies in the sky"

"A car drives"



Tokenized: [1 2 3 4 5 6]

Tokenized: [1 7 8]



Based on the **tokens** and **vectors**, propose the next word with **high probability**



"A car drives ... on the road"



1

DEMO



2

TAKING OFF WITH CODE

2 @AGENTS FOR SCOPE

@workspace

Navigate code in workspace by finding relevant files or classes.

@azure

Assistance with Azure-related topics or services

@github

Assistance with GitHub-related topics or services

@terminal

Assistance with how to run terminal commands

@vscode

Assistance with VS Code editor configuration

2

/SLASH COMMANDS

/explain

Explain code

/tests

Create unit tests

/fix

Propose a fix for problems in the selected code

/new

Scaffold a new project

/help

Get help on Copilot chat

2

#HASHTAG COMMANDS

#file

Specific File

#selection

Selected Code

#editor

“Visible” Source Code

#git

Git context

#terminalSelection

Selected Text in Terminal

#codebase

Search through the open codebase

2

PROMPT ENGINEERING

Prompt engineering is the process of designing and creating **high-quality prompts** that can be used to generate accurate and useful **code suggestions** with Copilot.

“The art of **coaxing** the **best possible result** out of a large language model”

Jennifer Marsman
Principal Engineer, Microsoft AI Research

2

BASIC TECHNIQUES

Zero-shot prompting

Write a C# program for a simple calculator that performs basic arithmetic operations based on user input.

One-shot and Few-shot prompting

Create a c# model for a FlightLogSignature property.

Example: 17121903-DEP-ARR-WB001

17th of December 1903 Departure from Kitty Hawk, NC A at Manteo, NC Flight number WB001

2

DEMO

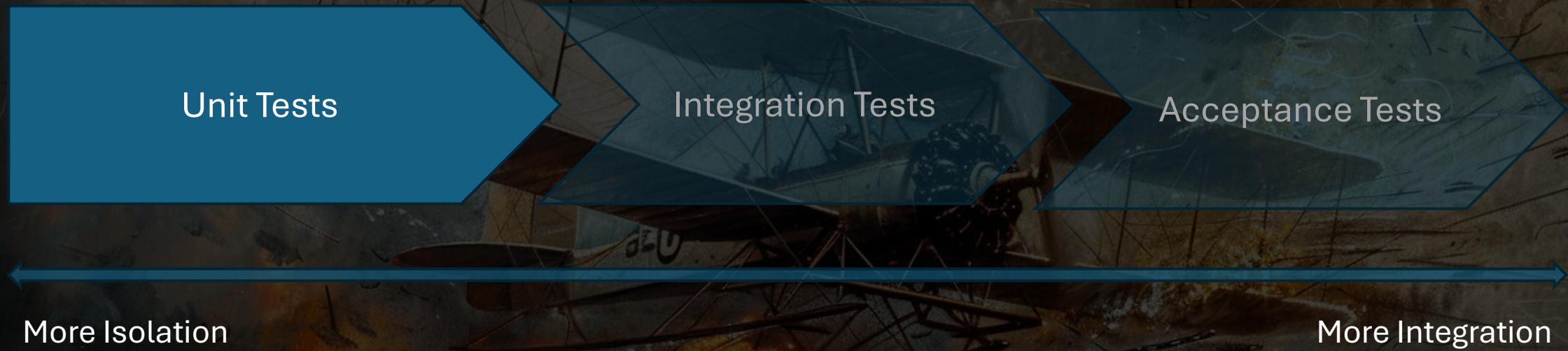


3

UNIT TESTING

3

UNIT TESTING



GitHub Copilot's design
excels at **handling smaller code** segments
to **maintain full context**

3

DEMO 3



ykhadas

Generate tests for FlightsController.cs



GitHub Copilot

Sure, I can help you generate some unit tests for your `FlightsController.cs`. However, without the actual code, I'll provide a general example of how you might structure your tests.

Assuming you have a `FlightsController` class with methods like `GetFlight(int id)`, `AddFlight(Flight flight)`, `UpdateFlight(int id, Flight flight)`, and `DeleteFlight(int id)`, here's how you might write unit tests for these methods:

Without context you will get “approximate” results

```
public class FlightsControllerTests
{
    private readonly FlightsController _controller;
    private readonly Mock<IFlightService> _serviceMock;

    public FlightsControllerTests()
    {
        _serviceMock = new Mock<IFlightService>();
        _controller = new FlightsController(_serviceMock.Object);
    }

    [Fact]
    public void GetFlight_WhenCalled_ReturnsFlight()
    {
        // Arrange
        var flight = new Flight { Id = 1, Name = "Test Flight" };
        _serviceMock.Setup(s => s.GetFlight(It.IsAny<int>())).Returns(flight);

        // Act
        var result = _controller.GetFlight(1);

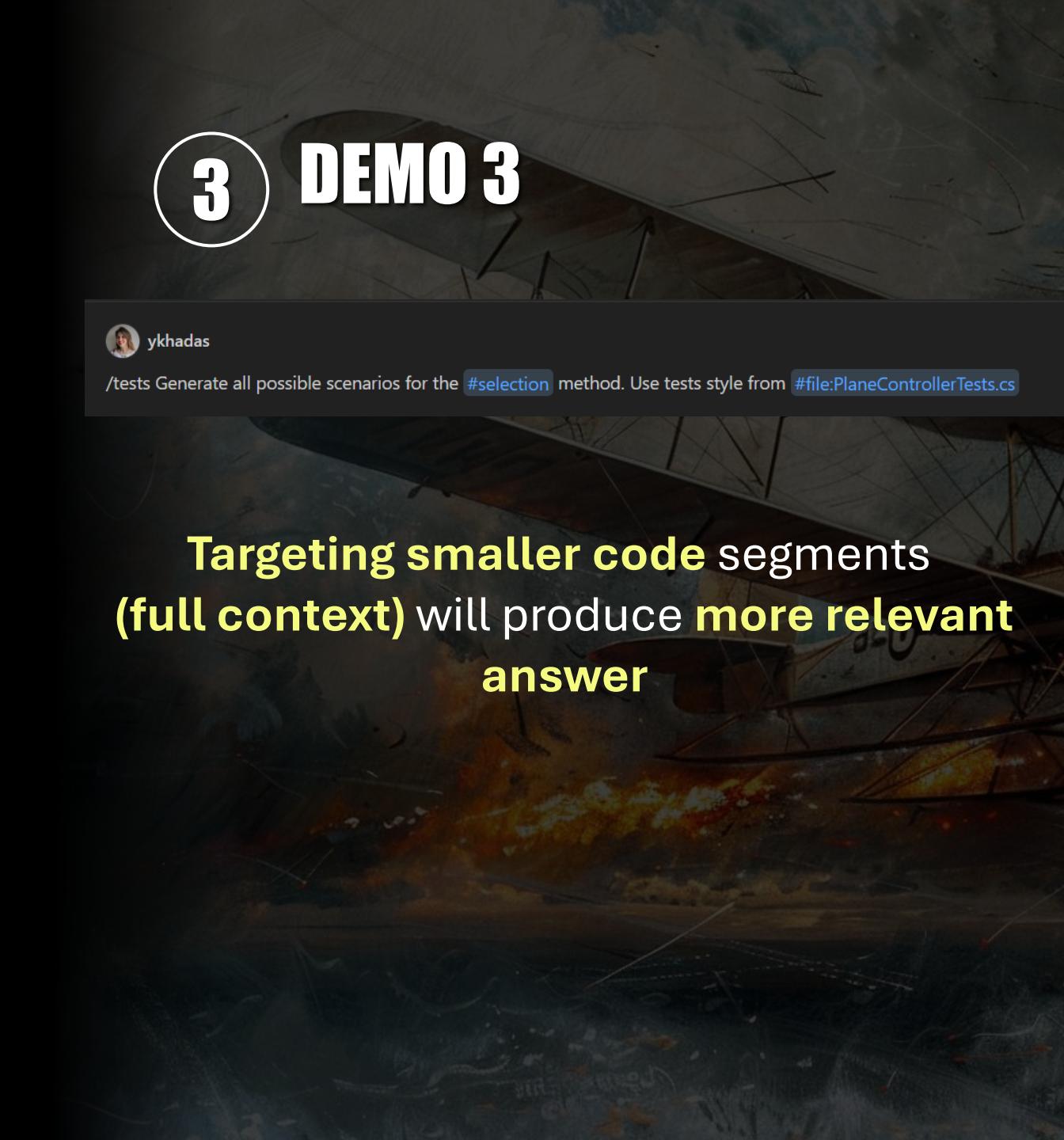
        // Assert
        Assert.IsType<OkObjectResult>(result.Result);
        Assert.Equal(flight, (result.Result as OkObjectResult).Value);
    }

    [Fact]
    public void AddFlight_WhenCalled_ReturnsFlight()
    {
        // Arrange
        var flight = new Flight { Name = "Test Flight" };
        _serviceMock.Setup(s => s.AddFlight(It.IsAny<Flight>())).Returns(flight);

        // Act
        var result = _controller.AddFlight(flight);

        // Assert
        Assert.IsType<CreatedAtActionResult>(result.Result);
        Assert.Equal(flight, (result.Result as CreatedAtActionResult).Value);
    }

    // Similarly, you can add tests for UpdateFlight and DeleteFlight
}
```

A black and white photograph of a Wright Brothers biplane in flight, viewed from below. The aircraft has two sets of wings and a single-seat cockpit.

3

DEMO 3



ykhadas

/tests Generate all possible scenarios for the `#selection` method. Use tests style from `#file:PlaneControllerTests.cs`

**Targeting smaller code segments
(full context) will produce more relevant
answer**

```
namespace WrightBrothersApi.Tests.Controllers
{
    public class FlightsControllerTests
    {
        private readonly ILogger<FlightsController> _logger;
        private readonly FlightsController _flightsController;

        public FlightsControllerTests()
        {
            _logger = Substitute.For<ILogger<FlightsController>>();
            _flightsController = new FlightsController(_logger);
        }

        [Fact]
        public void Search_ReturnsFlightWhenExists()
        {
            // Arrange
            var flightNumber = "Test Flight";

            // Act
            var result = _flightsController.Search(flightNumber);

            // Assert
            var okObjectResult = (OkObjectResult)result.Result!;
            var returnedFlight = (Flight)okObjectResult.Value!;
            returnedFlight.Should().NotBeNull();
            returnedFlight.FlightNumber.Should().Be(flightNumber);
        }

        [Fact]
        public void Search_ReturnsNotFoundWhenFlightDoesNotExist()
        {
            // Arrange
            var flightNumber = "Nonexistent Flight";

            // Act
            var result = _flightsController.Search(flightNumber);

            // Assert
            result.Result.Should().BeOfType<NotFoundResult>();
        }
    }
}
```

3

UNIT TESTING

unit test
scenarios for
#selection
and the tests
should
match **#file**

The screenshot shows a code editor interface with a dark theme. At the top, a tab bar indicates the file is `PlaneControllerTests.cs`. Below the tabs, a breadcrumb navigation shows the project structure: `wrightBrothersApi > WrightBrothersApi > Controllers > C# PlanesController.cs`. The main area displays the `GetById` method from `PlanesController.cs`:

```
5     {  
9         {  
52             *  
53             [HttpGet("{id}")]  
54             public ActionResult<Plane> GetById(int id)  
55             {  
56                 var plane = Planes.Find(p => p.Id == id);  
57                 if (plane == null)  
58                 {  
59                     return NotFound();  
60                 }  
61                 return Ok(plane);  
62             }  
63         }  
64     }
```

A yellow circle with the number 3 is positioned above the tab bar. A yellow circle with the number 1 is positioned over the first line of the `GetById` method. A yellow circle with the number 2 is positioned over the text in a tooltip at the bottom left.

CHAT

GitHub Copilot

Hi @thijslimmen, how can I help you?

Generate all unit test scenarios for **#selection** and the tests should match **#file**: 2



3 DEMO



4 TROUBLESHOOTING

4

TROUBLESHOOTING

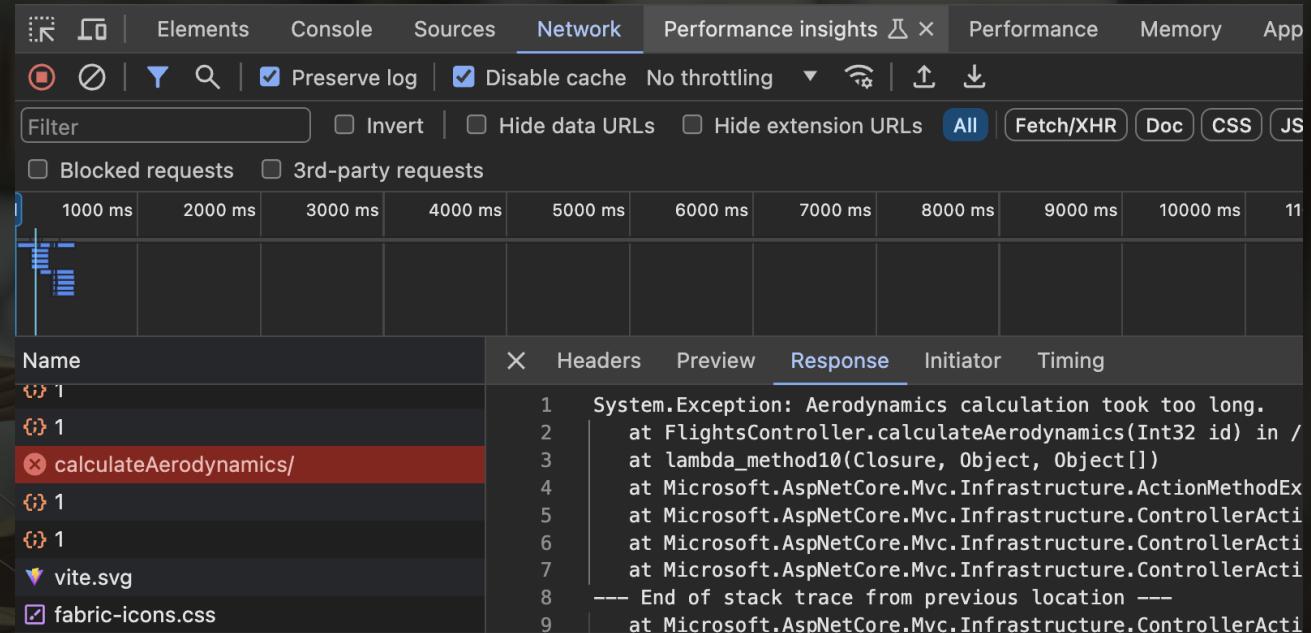
GitHub Copilot is
useful for **debugging**
and **troubleshooting**:

- Exception **explanation**
- Code **analysis**
- Suggesting **fixes**
- Code **refactoring**
- Code **documentation**
- Helps with automated **testing**

4

IMPROVE PERFORMANCE

DEMO 1



Found 25997 prime numbers.

Elapsed Time: 4.688 seconds

fail: Microsoft.AspNetCore.Diagnostics.DeveloperExceptionPageMiddleware[1]

An unhandled exception has occurred while executing the request.

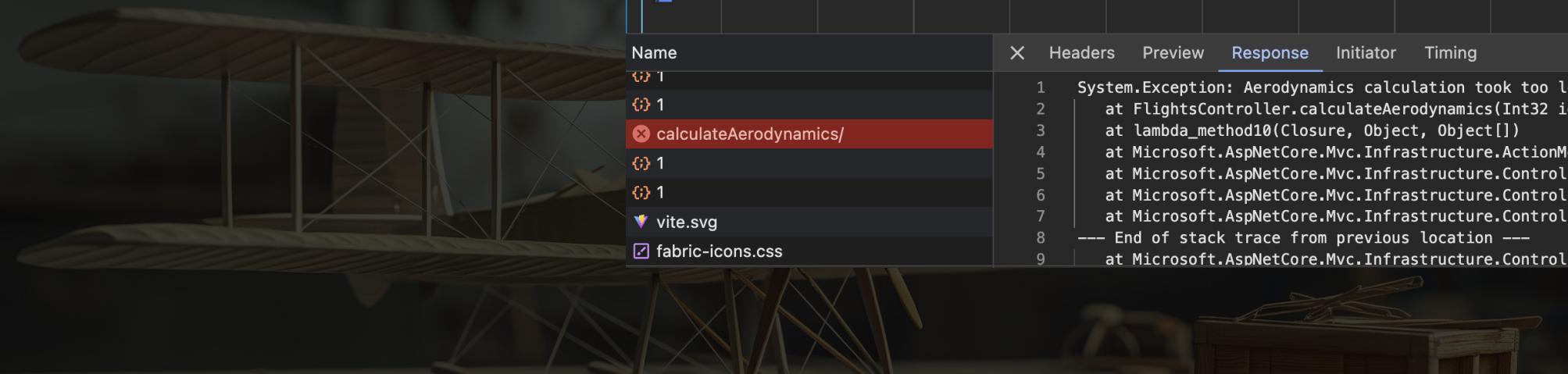
System.Exception: Aerodynamics calculation took too long.

at FlightsController.calculateAerodynamics(Int32 id) in /Users/thijslimmen/Documents/Projects/Xpirit/copilot-beyond-basics/WrightBrothersApi/WrightBrothersApi/Controllers/FlightsController.cs:line 212

4

IMPROVE PERFORMANCE

DEMO 1



Found 25997 prime numbers.

Elapsed Time: 4.688 seconds

fail: Microsoft.AspNetCore.Diagnostics.DeveloperExceptionPageMiddleware[1]

An unhandled exception has occurred while executing the request.

System.Exception: Aerodynamics calculation took too long.

at FlightsController.calculateAerodynamics(Int32 id) in /Users/thijslimmen/Docu
ments/Projects/Xpirit/copilot-beyond-basics/WrightBrothersApi/WrightBrothersApi/Controll
ers/FlightsController.cs:line 212

4

IMPROVE PERFORMANCE DEMO 1

The screenshot shows a terminal window with several tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PO. A large yellow arrow points from the GitHub Copilot analysis on the left to the terminal output on the right.

GitHub Copilot Analysis:

- User message: "#terminalSelection what is wrong with the performance? #file:FlightsController.cs"
- GitHub Copilot response:
 - Used 2 references >
 - The performance issue is in the `calculateAerodynamics` method of the `FlightsController` class. This method calculates prime numbers between 2 and 300000, which is a computationally expensive operation, especially with the

Terminal Selection:

```
pl.Invoke(HttpContext context)
info: FlightsController[0]
GET >>> 1 >>>
info: FlightsController[0]
GET >>> 1 >>>
Found 25997 prime numbers.
Elapsed Time: 4.688 seconds
fail: Microsoft.AspNetCore.DiagnosticsDeveloperExceptionPageMiddleware: An unhandled exception has occurred while executing the request.
System.Exception: Aerodynamics calculation took too long.
   at FlightsController.calculateAerodynamics(IList<Flight> flights)
   at lambda_method10(Closure closure, Object object)
   at Microsoft.AspNetCore.Mvc.Infrastructure.ActionInvoker
```

4

TROUBLESHOOTING

Copilot assists with **refactoring** code to improve **readability**, **structure**, and **maintainability**

The screenshot shows a GitHub Copilot interface integrated into a code editor. A yellow circle labeled '2' highlights a tooltip suggesting to "Refactor the selected code to make it more readable and maintainable." Another yellow circle labeled '1' highlights the code editor where GitHub Copilot has provided several refactoring suggestions, such as using methods for switch statements and constants for error messages.

CHAT: GITHUB COPILOT + ⌂ ...

Refactor the selected code to make it more readable and maintainable.

GitHub Copilot

Used 1 reference >

Sure, I can suggest a few improvements to make the code more readable and maintainable:

1. Use a method to encapsulate the logic of each case in the switch statement.
2. Use constants or an enum for the error messages.
3. Use a method to find the flight.

Here's the refactored code:

```
[HttpPost("{id}/status")]
public ActionResult UpdateFlightStatus(int id, FlightStatus newStatus)
```

How can I improve the performance of my code?

Ask Copilot or type / for commands ➤

C# FlightsController.cs X

WrightBrothersApi > Controllers > C# FlightsController.cs > FlightsController >

```
8 public class FlightsController : ControllerBase
88
89     [HttpPost("{id}/status")]
90     public ActionResult UpdateFlightStatus(int id, FlightStatus
91     {
92         var flight = Flights.Find(f => f.Id == id);
93         if (flight != null)
94         {
95             switch (newStatus)
96             {
97                 case FlightStatus.Booking:
98                     if (DateTime.Now > flight.DepartureTime)
99                     {
100                         return BadRequest("Cannot board, past departure time");
101                     }
102                     break;
103
104                 case FlightStatus.Departed:
105                     if (flight.Status != FlightStatus.Booking)
106                     {
107                         return BadRequest("Flight must be in Booking status to depart");
108                     }
109                     break;
110
111                 case FlightStatus.InAir:
112                     if (flight.Status != FlightStatus.Departed)
113                         return BadRequest("Flight must be in Departed status to be in air");
114                     break;
115             }
116         }
117     }
```



4

DEMO

ADVANCED



5

CHAIN OF THOUGHT

5

CHAIN OF THOUGHT

Enables **Complex Reasoning** capabilities through **Intermediate Reasoning** steps to get **Better Results** on **complex tasks**

5

CHAIN OF THOUGHT

AEROBATICS SEQUENCE SIGNATURE (FICTIONAL)

Parse a AerobaticSequenceSignature property into a c# model.

AerobaticSequence Examples

- L4B-H2C-R3A-S1D-T2E
- L1A-H1B-R1C-T1E
- L2A-H2B-R2C

Maneuver

- Manouvers: L = Loop, H = Hammerhead, R = Roll, S = Spin, T = Tailslide
- Number represents repeat count
- The Letter represents difficulty (A-E)
- Difficulty multipliers: A = 1.0, B = 1.2, C = 1.4, D = 1.6, E = 1.8

5

CHAIN OF THOUGHT

AEROBATICS SEQUENCE SIGNATURE (FICTIONAL)

Parse a AerobaticSequenceSignature property into a c# model.

AerobaticSequence Examples

- L4B-H2C-R3A-S1D-T2E
- L1A-H1B-R1C-T1E
- L2A-H2B-R2C

Maneuver

- Manouvers: L = Loop, H = Hammerhead, R = Roll, S = Spin, T = Tailslide
- Number represents repeat count
- The Letter represents difficulty (A-E)
- Difficulty multipliers: A = 1.0, B = 1.2, C = 1.4, D = 1.6, E = 1.8

5

CHAIN OF THOUGHT

CALCULATE DIFFICULTY

```
## AerobaticSequence Difficulty Method
```

- Add a difficulty calculation method with the following rules:
- A roll after a loop is scored double
- A spin after a tailslide is scored triple

5

CHAIN OF THOUGHT

CHAIN OF THOUGHT REASONING

Chain-of-Thought reasoning

Example: L4B-R3A-H2C-T2E-S1D

- Loop: $4 * 1.2 = 4.8$
- Roll: $3 * 1 * 2$ (roll after a loop) = 6.0
- Hammerhead: $2 * 1.4 = 2.8$
- Tailslide: $2 * 1.8 = 3.6$
- Spin: $1 * 1.6 * 3$ (spin after a tailslide) = 4.8

Total: 22

5

CHAIN OF THOUGHT

CHAIN OF THOUGHT REASONING

Chain-of-Thought reasoning

Example: L4B-R3A-H2C-T2E-S1D

- Loop: $4 * 1.2 = 4.8$
- Roll: $3 * 1 * 2$ (roll after a loop) = 6.0
- Hammerhead: $2 * 1.4 = 2.8$
- Tailslide: $2 * 1.8 = 3.6$
- Spin: $1 * 1.6 * 3$ (spin after a tailslide) = 4.8

Total: 22

5

CHAIN OF THOUGHT

CHAIN OF THOUGHT REASONING

Chain-of-Thought reasoning

Example: L4B-R3A-H2C-T2E-S1D

- Loop: $4 * 1.2 = 4.8$
- Roll: $3 * 1 * 2$ (roll after a loop) = 6.0
- Hammerhead: $2 * 1.4 = 2.8$
- Tailslide: $2 * 1.8 = 3.6$
- Spin: $1 * 1.6 * 3$ (spin after a tailslide) = 4.8

Total: 22

Parse a AerobaticSequenceSignature property into a c# model.

AerobaticSequence Examples

- L4B-H2C-R3A-S1D-T2E
- L1A-H1B-R1C-T1E
- L2A-H2B-R2C

Maneuver

- Manouvers: L = Loop, H = Hammerhead, R = Roll, S = Spin, T = Tailslide
- Number represents repeat count
- The Letter represents difficulty (A-E)
- Difficulty multipliers: A = 1.0, B = 1.2, C = 1.4, D = 1.6, E = 1.8

AerobaticSequence Difficulty Method

- Add a difficulty calculation method with the following rules:
- A roll after a loop is scored double
- A spin after a tailslide is scored triple

Chain-of-Thought reasoning

Example: L4B-R3A-H2C-T2E-S1D

- Loop: $4 * 1.2 = 4.8$
- Roll: $3 * 1 * 2(\text{roll after a loop}) = 6.0$
- Hammerhead: $2 * 1.4 = 2.8$
- Tailslide: $2 * 1.8 = 3.6$
- Spin: $1 * 1.6 * 3(\text{spin after a tailslide}) = 4.8$

Total: 22

```
public class AerobaticSequence
{
    9 references
    public List<Maneuver> Maneuvres { get; set; } = new List<Maneuver>();
    1 reference
    public double Difficulty { get; set; }

    0 references
    public static AerobaticSequence Parse(string signature)
    {
        var sequence = new AerobaticSequence();
        var maneuvers = Regex.Matches(signature, @"(?<=[A-Z]\d+[A-E])");
        foreach (Match maneuver in maneuvers)
        {
            var type = maneuver.Value[0].ToString();
            var repeatCount = int.Parse(maneuver.Value.Substring(1, maneuver.Value.Length - 2));
            var difficulty = maneuver.Value[^1];
            sequence.Maneuvres.Add(new Maneuver { Type = type, RepeatCount = repeatCount, Difficulty = difficulty });
        }
        sequence.CalculateDifficulty();
        return sequence;
    }

    1 reference
    private void CalculateDifficulty()
    {
        double difficulty = 0;
        for (int i = 0; i < Maneuvres.Count; i++)
        {
            var multiplier = GetMultiplier(Maneuvres[i].Difficulty);
            var repeatCount = Maneuvres[i].RepeatCount;
            if (i > 0 && (Maneuvres[i - 1].Type == "L" && Maneuvres[i].Type == "R" || Maneuvres[i - 1].Type == "T" && Maneuvres[i].Type == "S"))
            {
                multiplier *= (Maneuvres[i].Type == "R" ? 2 : 3);
            }
            difficulty += repeatCount * multiplier;
        }
        Difficulty = Math.Round(difficulty, 2);
    }

    1 reference
    private double GetMultiplier(char difficulty)
    {
        return difficulty switch
        {
            'A' => 1.0,
            'B' => 1.2,
            'C' => 1.4,
            'D' => 1.6,
            'E' => 1.8,
            _ => throw new ArgumentException("Invalid difficulty"),
        };
    }
}
```

HTTP/1.1 201 Created
Connection: close
Content-Type: application/json; charset=utf-8
Date: Wed, 03 Apr 2024 21:17:19 GMT
Server: Kestrel
Location: <http://localhost:1903/Flights/4>
Transfer-Encoding: chunked

```
{  
    "id": 4,  
    "flightNumber": "WB004",  
    "aerobaticSequenceSignature": "L4B-R3A-H2C-T2E-S1D",  
    "aerobaticSequence": {  
        "difficulty": 22  
    }  
}
```

5

WAYS

CHAIN OF THOUGHT

"Let's think step by step"
(Zero-shot CoT)

5

CHAIN OF THOUGHT

More prompt
engineering techniques:

<https://www.promptingguide.ai>





6

AUTOMATION PIPELINES

Chain-of-Thought prompting for what needs to be done and how to do it.



PagelesR

"Let's set up a CI/CD pipeline for a .NET 7 Web API application, which includes stages for Build, Infrastructure as Code (IaC) deployment using Bicep, and Quality Assurance (QA). Each stage should be clearly defined, and necessary files and configurations must be scaffolded out.

1. Build Stage for .NET 7 Web API:

- The application is built with .NET 7, utilizing C# as the programming language. Scaffold the build process, which should include steps for restoring NuGet packages, compiling the solution, running unit tests, and publishing the output. Create a 'build.yaml' file that outlines these steps for the CI pipeline.

2. IaC Stage with Bicep Execution:

- For deployment to Azure, identify the required Azure resources (e.g., Azure App Service for hosting the Web API, Azure SQL Database for data storage). Scaffold Bicep templates for these resources, ensuring they're modular and reusable.
- Create a 'main.bicep' file that acts as the entry point, invoking other Bicep files (e.g., 'appservice.bicep', 'sqldatabase.bicep') to define the entire infrastructure. Include a step in the CI/CD pipeline ('iac-deploy.yaml') to authenticate with Azure and execute 'main.bicep', deploying the infrastructure.

3. QA Stage:

- Detail the process for deploying the .NET 7 Web API to the provisioned Azure App Service in a QA environment. This includes applying database migrations if necessary and running integration and acceptance tests.
- Scaffold a 'qa-deploy.yaml' file that outlines the steps for deploying the application to Azure, setting up the environment, and executing tests.

In addition to scaffolding these files, provide explanations for each file's purpose and the critical sections within them. This approach ensures a comprehensive understanding and correct implementation of the pipeline for deploying a .NET 7 Web API application to Azure."



GitHub Copilot

Sure, let's break down the process and scaffold the necessary files.

6

AUTOMATION PIPELINES

Chain-of-Thought prompting for what needs to be done and how to do it.

1



GitHub Copilot Generating



Sure, let's start by creating the necessary files and configurations for each stage of the CI/CD pipeline.

1. Build Stage for .NET 7 Web API:

The 'build.yml' file will be used to define the build process for the .NET 7 Web API application. This includes steps for restoring NuGet packages, compiling the solution, running unit tests, and publishing the output.

```
# build.yml
name: Build

on:
  push:
    branches:
      - main
```

2

2. IaC Stage with Bicep Execution:

The 'main.bicep' file will be used to define the infrastructure for the Azure resources required by the application. The 'iac-deploy.yml' file will be used to authenticate with Azure and execute the 'main.bicep' file, deploying the infrastructure.

3

3. QA Stage:

The 'qa-deploy.yml' file will be used to deploy the .NET 7 Web API to the provisioned Azure App Service in a QA environment. This includes steps for setting up the environment and executing tests.

4

4. Chain Workflows:

The 'workflow_run' event is used to chain the workflows together. The Build workflow triggers the IaC workflow, which then triggers the QA workflow. This ensures that each stage of the pipeline is executed in the correct order.

The background of the image depicts a dramatic aerial dogfight between two World War I era biplanes. One plane is silhouetted against a bright explosion or fire in the lower left foreground. Another plane is visible in the upper left, and a third is partially visible on the right. The sky is filled with smoke, fire, and the silhouettes of many other planes, creating a chaotic and intense atmosphere.

5

DEMO

A detailed wooden model of a vintage biplane is the central focus, resting on a light-colored wooden workbench. The model features two sets of wings, a propeller at the front, and a tail section. In the background, there's a large wooden barrel and various tools and equipment, suggesting a workshop or hobby room environment.

6

UI/UX

6

ADD BUTTON UI ELEMENT

The image shows a mobile application screen with a light orange background. At the top, there is a yellow circle icon on the left and a dark grey triangle icon on the right. The title "Dawn of Aviation" is centered in large white font. Below the title is a subtitle: "Journey back to where it all began with the Wright Brothers' historic flights." In the center, there is a detailed illustration of a Wright Flyer biplane. Below the illustration, there are three rectangular cards, each featuring a small version of the same biplane and its name: "Wright Flyer", "Wright Flyer II", and "Wright Model A". At the bottom of the screen, there is a large, rounded rectangular button with a brown gradient background and white text that reads "ADD NEW PLANE BUTTON HERE".

Dawn of Aviation

Journey back to where it all began with the Wright Brothers' historic flights.

Wright Flyer

Wright Flyer II

Wright Model A

ADD NEW PLANE
BUTTON HERE

6

FLIGHT DETAILS

Wright Flyer

1903

The first successful heavier-than-air powered aircraft.



6

DEMO 1



thijslimmen

@workspace Create a button component and route the button to the add new plane page

Design

- Create a button that is in the same style as [#file:PlaneList.tsx](#)
- Add a plus icon to the button
- Spacing on the top and bottom of the button
- Elevate the button when hovered

Technical Requirements

- Create a new button component
- use [@heroicons/react/24/solid](#) for the plus icon

Result

```
import React from "react";
import { PlusIcon } from "@heroicons/react/24/solid";
import { useNavigate } from "react-router-dom";

const AddPlaneButton: React.FC = () => {
  const navigate = useNavigate();

  const handleClick = () => {
    navigate("/new-plane");
  };

  return (
    <button
      onClick={handleClick}
      className="mt-3 mb-3 bg-amber-100 hover:bg-amber-200 rounded-lg shadow-lg hover:shadow-xl transition duration-200 ease-in-out"
    >
      <PlusIcon className="h-6 w-6" />
    </button>
  );
}

export default AddPlaneButton;
```

6

DEMO 1



Wright Flyer



Wright Flyer II



Wright Model A

+ Add Plane

6

DEMO 2



thijslimmen

Create a Form inside the page at `#editor`

Design

- Apply tailwind classes to match the design of `#file:PlaneList.tsx`

Technical Requirements

- Create fields based on file `#file:Plane.ts`
- Formik for form handling
- Yup for validations

Editor

NewPlane.tsx

```
WrightBrothersFrontend > src > pages > NewPlane.tsx > ...  
1 import Container from "../components/Container";  
2  
3 const NewPlane = () => {  
4   return (  
5     <Container>  
6       <h1>New Plane</h1>  
7     </Container>  
8   );  
9 };  
10  
11 export default NewPlane;
```

```
export interface Plane {  
  id: number;  
  name: string;  
  year: number;  
  description: string;  
  rangeInKm: number;  
}
```

6

DEMO 2

New Plane

ID

Name _____

Year

Required

Description

Description

Range in Km

Required

Submit

6

DEMO 3



thijslimmen

@workspace /new Playwright UI Tests

Test

1. Navigate to "/new-plane"
2. Fill out the form with attribute "name": name, year, description, rangeInKm

Technical Requirements

- Headless is False
- localhost:5173 is the base URL
- Typescript
- Include package.json
- Use @playwright/test library

Make a complete solution

6

DEMO 3



GitHub Copilot used @workspace /new

Sure, here's a proposed directory structure for a Playwright UI Tests project:

```
✓ src
  ✓ tests
    TS new-plane.test.ts
  ✓ utils
    TS setup.ts
    TS playwright.config.ts
  { } package.json
  ⓘ README.md
```

Create Workspace

6

DEMO 3

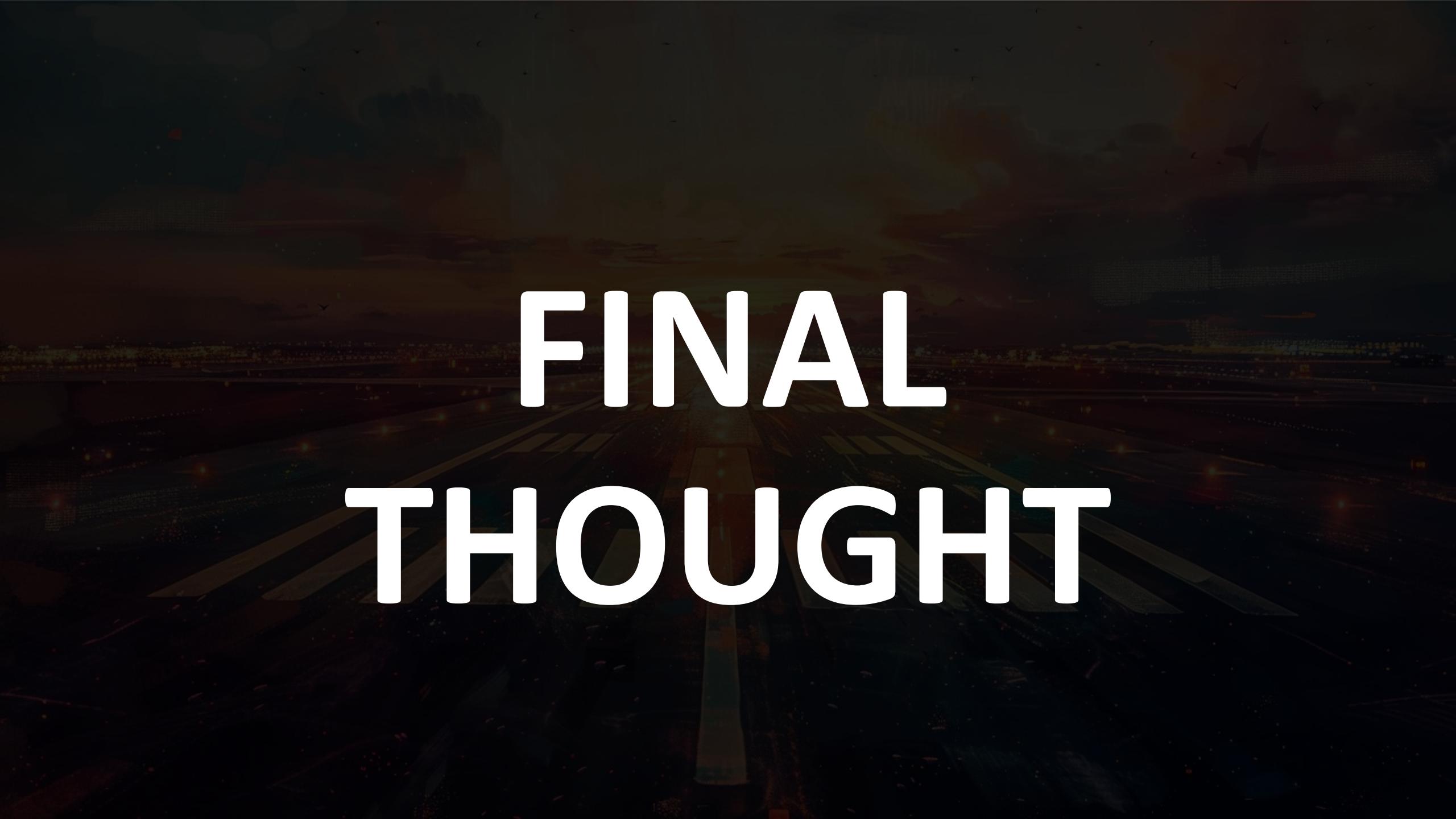
```
→ playwright-ui-tests git:(main) ✘ npm run test
```

```
> playwright-ui-tests@1.0.0 test
> playwright test
```

Running 1 test using 1 worker

```
✓ 1 src/tests/new-plane.test.ts:3:5 > New Plane Form Test (899ms)
```

```
1 passed (2.0s)
```



**FINAL
THOUGHT**



FINAL THOUGHTS

FINAL THOUGHTS

Do's

- Set clear goals
- Provide detailed instructions
- Refine through feedback
- Use for creativity and exploration
- Make flight hours by using the tool

Don'ts

- Rely blindly; verify
- Expect instant perfection



FINAL THOUGHTS

FINAL THOUGHTS

Copilot **predicts**
code suggestions
based on **data** it's
trained on

- Suboptimal Suggestions
 - May be using outdated LLM version
 - Struggles with niche or new technologies
 - May oversimplify complex issues
 - Risks including deprecated approaches
- Need for Human Judgement
 - Copilot is not a replacement for developer insights
- Quality Assurance Pipelines
 - Pull Requests and Code Reviews
 - Static Code Analysis
 - Automated Testing
 - Monitoring and Observability



FINAL THOUGHTS

GET STARTED WITH COPILOT

Quick Start:

<https://docs.github.com/en/copilot/quickstart>



Copilot Adventures:

<https://microsoft.github.io/CopilotAdventures/>



Copilot Clips



More about Xebia
www.xebia.com