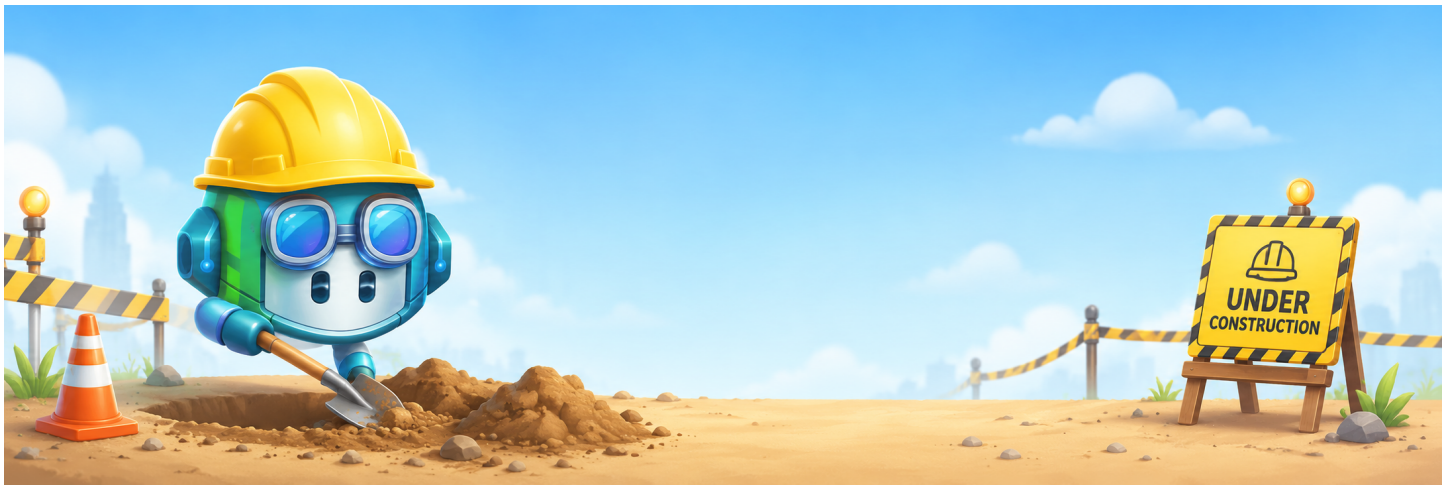
GitHub Copilot Training Material

AFRY Training Material

AFRY IT

AFRY

Table of contents



1. GitHub Copilot - Training Material

Training material for GitHub Copilot for everyone - from first login to building and deploying full applications.

1.1 General

- What is GitHub Copilot
- Security, Trust & Compliance
- Getting Access
- Costs and License Model
- Copilot Ambassadors

1.2 Getting Started

- Software You Need
- Overview - 🎬 Video
- Spec-driven Development
- System Instructions and README
- Tools and MCP
- Skills
- Agents

1.3 AFRY Tools

- Brave Copilot
- MCP Servers

1.4 Vibe Coding

- GitHub and Git
- Introduction
- Prerequisites
- Create Repository - 🎬 Video
- Create a Web App - 🎬 Video
- Deploy to Azure - 🎬 Video
- Add Authentication - 🎬 Video
- Store Data - 🎬 Video
- Talk to an LLM - 🎬 Video

- glossary.md - terminology and definitions

2. General

2.1 What is GitHub Copilot?

GitHub Copilot is an AI that reads your entire project and understands intent. It writes, plans, executes, fixes errors, and iterates autonomously. It connects to live data and external systems via MCP tools, and it works in conversation: you describe a goal, it works towards it. GitHub Copilot lives in your code editor, and runs locally in your computer.

2.1.1 Who is it for?

Copilot is useful for anyone who has repetitive tasks, documents to write, data to process, or ideas to build.

Developer

- Write code, fix bugs, generate tests
- Review pull requests, document APIs
- Build full web apps with Agent mode

Engineer

- Create plugins and scripts for engineering software
- Transform data - eg. GIS to BIM
- Automate runs, calculations. Generate VBA, Python, etc.

Analyst / Data

- Write Python or SQL scripts without memorising syntax
- Query databases and generate formatted reports
- Automate repetitive Excel or data-prep tasks

Project Manager / Business Analyst

- Draft documents, plans, and user stories
- Summarise meeting notes and sprint data
- Create structured requirement specs from rough notes

Anyone at AFRY

- Draft and send status emails - auto-written from real data
- Create AFRY-branded PowerPoint decks from bullet points
- Find the right colleague or expert - AI searches the company directory
- Pull data directly from AFRY's tenant, without leaving VS Code

---

2.1.2 What you can build or get done

1. **Build a web app from an idea** - React + Azure, deployed with CI/CD, backed with a database and with AFRY login
2. **Draft and send a status email** - Copilot reads sprint data, writes the email, sends it via AFRY's mail tools
3. **Create a branded slide deck** - describe your content, Copilot builds the AFRY-branded .pptx
4. **Analyse application logs** - ask Copilot to run KQL against Azure Application Insights
5. **Write a data analysis script** - no prior Python experience needed; describe what you want
6. **Generate images for presentations** - natural language to image, delivered as PNG or slide image
7. **Automate a repetitive Excel workflow** - Copilot writes and runs the script

---

2.1.3 Three modes: Ask, Plan, Agent

Ask

Have a conversation with your codebase, your files, or the web. Ask questions, get explanations, explore options. Nothing changes - pure Q&A.

Plan

Tell Copilot what you want to achieve. It researches the problem and writes a step-by-step implementation plan. No files are touched - you read and approve before anything happens.

Agent

Copilot takes the plan and executes it. It reads files, writes code, runs terminal commands, fixes errors, and iterates. You stay in control - approve sensitive actions before they run.

2.1.4 Pick the right model for the job

GitHub Copilot Enterprise gives you access to all leading AI models - switch in one click.

Model	Strength	Best for
Auto	Automatically choose the cheapest model	Gives you the fastest and cheapest option at a 10% discount
GPT-5.4	Speed + broad knowledge	Everyday chat, quick edits, inline completions
Claude Haiku 4.5	Fast, cheap, intelligent	For everyday tasks
Claude Sonnet 4.6	Good value for money, nuanced reasoning	Large codebases, long documents, careful planning
Claude Opus 4.6	Long context,	Large codebases, planning
Gemini 3.5 Flash	Multi-modal, fast	Images, data analysis, code combined
GPT-5.4 mini	Fast and lightweight	Simple Q&A, high-volume tasks

---

2.1.5 Connected to AFRY

Out of the box, Copilot can chat about anything, read and edit your files, write and run code, and browse the web.

AFRY's tools extend this further via MCP servers

- **Look up any AFRY colleague** - full profile, manager chain, direct reports
- **Create and edit AFRY-branded PowerPoint slides**
- **Send emails to your team** - auto-composed from data
- **Deep research**, let GitHub Copilot search for material
- **Query Application Insights logs** with natural language (KQL)
- **Peek at Azure Service Bus queues** - dead-letter and active
- **Generate images** - into slides or files

2.1.6 Teach Copilot how you work - Skills

Out of the box, Copilot is a generalist. A skill is a plain-text instruction file that makes it a specialist.

You write the skill once - the steps, the conventions, the tools - and anyone on the team can invoke it just by describing what they want.

Examples at AFRY:

- Build a slide deck that follows the AFRY brand guide
- Build a web app and deploy it in Azure
- Apply AFRY brand guidelines to a presentation
- Have Copilot write text as a human and minimize signs of AI slop

---

2.1.7 Start today

GitHub Copilot is available to all AFRY employees.

Three steps

1. **Order VS Code + GitHub Copilot** from the AFRY Application Kiosk
2. **Sign in to AFRY's GitHub Enterprise** - follow the prompts in VS Code
3. **Open Copilot Chat** and ask your first question - try: "*What can you help me with today?*"

2.2 Security, Trust & Compliance

2.2.1 Is GitHub Copilot safe to use on client projects?

GitHub Copilot Business and Enterprise can be used for professional work when accessed through AFRY-managed accounts and in line with AFRY and client data-handling rules. The enterprise agreement with GitHub provides contractual data protection - not just a policy promise.

Key protections under the agreement:

- **AFRY-managed identity and access** - accounts are provisioned through AFRY IT; access is controlled via SSO, with no personal accounts
- **No model training on your code or prompts** - GitHub states that Business and Enterprise customer data is not used to train AI models, under the Data Processing Agreement
- **All data is encrypted** in transit and at rest
- **Covered by a Data Processing Agreement (DPA)** with GitHub, including EU Standard Contractual Clauses

Copilot prompts and context are processed by GitHub Copilot and its model-hosting infrastructure. Where configured, GitHub Copilot data residency can keep supported processing within the selected EU or US geography - but this is not the same as data staying inside AFRY's own environment.

2.2.2 Certifications

GitHub Copilot has its own compliance reports and certifications, separate from Azure or Microsoft 365.

Certification	What it covers
SOC 2 Type II	Security, availability, and confidentiality controls - externally audited
ISO 27001:2022	International standard for information security management
ISO 42001:2023	AI-specific management standard - relevant to EU AI Act readiness
GDPR / DPA	EU Standard Contractual Clauses for cross-border data transfers; EU data residency available

Compliance reports and certificates are available at copilot.github.trust.page and from your GitHub enterprise settings under Compliance.

2.2.3 What the enterprise agreement gives you

The agreement is what separates GitHub Copilot from personal or free-tier AI tools. In practice it means:

- **AFRY controls which features are enabled** and for whom - governance is built in from day one
- **Centralised audit log** - governance events, policy changes, and agent activity are recorded
- **SSO managed by AFRY IT** - no personal accounts, no blurring between personal and professional use
- **Data deletion on request** - subject to GitHub DPA terms
- **Single point of accountability** - GitHub under a signed contract

The Copilot audit log records governance and agent events (policy changes, seat assignments, agent activity). It does not include the content of local Copilot prompts or coding sessions.

2.2.4 Source

- GitHub Trust Center
- GitHub Data Protection Agreement
- GitHub Copilot Enterprise documentation

2.3 Getting Access

2.3.1 Step 1 - Request GitHub Copilot via the Application Kiosk

GitHub Copilot requires a licence with **manager approval**.

1. Go to the AFRY Application Kiosk
2. Search for and select **GitHub Copilot**
3. Submit the request and wait for manager approval before continuing

2.3.2 Step 2 - Install VS Code

VS Code is free to download. Admin rights are required - launching the installer triggers a ServiceNow request automatically.

- Download from https://code.visualstudio.com/
- Use default settings; check **"Add to PATH"** during installation

2.3.3 Step 3 - Sign in to GitHub Copilot

1. Open VS Code
2. Click the **GitHub Copilot icon** in the bottom-right corner
3. Click **Use AI features** - your browser will open for authentication
4. Your GitHub Enterprise username is in the email you received when registering (derived from your AFRY email)
5. Enter your username - the password field will be disabled; you are taken to AFRY's login page

2.3.4 Step 4 - Verify your setup

- Click the **chat icon** at the top of VS Code (or press `Ctrl+Alt+I`)

- Ask a simple question - e.g. "How do I create a Python function?"
- Click the **model dropdown** and confirm you have access to premium models (Claude, GPT, Gemini)

2.3.5 Also install

Priority	Tool	Purpose
🔴 Essential	VS Code	Your code editor
🔴 Essential	GitHub Copilot	Request via Application Kiosk
🟡 Recommended	Git	Version control
🟡 Recommended	Python	Scripting + required by many MCP tools
🟡 Recommended	Node.js	Required for MCP servers and web tools
⚪ Optional	GitHub CLI (`gh`)	Only needed for publishing websites
⚪ Optional	Azure CLI (`az`)	Only needed for Azure deployment

Full installation instructions including verification steps are available here: [[installation-instructions]]

2.4 Costs & License Model

2.4.1 How you get access at AFRY

When you request GitHub Copilot from the Application Kiosk, your section is billed monthly. You receive a **Copilot Business** licence by default.

To upgrade to a **Copilot Enterprise** licence (which includes additional features), contact the GitHub admins. *(Link to be provided.)*

2.4.2 The billing model

GitHub Copilot uses **GitHub AI Credits** for usage-based billing.

What is included in the seat price:

- Seat pricing - Copilot Business is $19/user/month
- Code completions and Next Edit Suggestions are **unlimited** and do not consume credits

How AI Credits work:

- All other features (chat, agent mode, code review) consume AI Credits
- Credits are spent based on **token usage** - input, output, and cached tokens - at the published rate for each model
- Unused credits **do not roll over** - they reset each billing cycle

---

2.4.3 What is a token?

A token is a small chunk of text - roughly ¾ of a word. Every message you send and every response you receive is measured in tokens.

The same task costs very different amounts depending on the model and the type of work:

Task	Token use	Credit impact
Inline code completion	Very low	Unlimited - no credits consumed
Quick chat question	Low	Minimal
Code review of a PR	Medium	Moderate
Agent session - implement a feature	High	Significant
Long agentic session with large context	Very high	Can be substantial

Heavier models cost more per token than lighter ones. Choosing the right model for the task matters.

⚠️ *The figures below are estimates based on published API rates and are not confirmed.*

How far does a Business seat ($19/month = 1,900 AI Credits) go?

One AI Credit = $0.01. The credits are spent at each model's published token rate:

Model	Blended cost per million tokens	Tokens covered by 1,900 credits
Claude Sonnet	~$6.00	~3.2M tokens
GPT-5 (standard)	~$3.44	~5.5M tokens
Claude Opus (heavy)	~$10.00	~1.9M tokens

What does that mean in practice?

A typical agent-mode session - with repository context, tool calls, reasoning, and edits - consumes roughly **100K–500K tokens**.

At that rate, **10–20 agent sessions on a heavy model can exhaust a full month's allowance in a day**.

Quick chat questions and code completions are far cheaper; it is sustained agentic work that drives the cost.

2.4.4 For Copilot Business (AFRY's plan)

- **$19/user/month** - includes **$19 in monthly AI Credits** per user
- Admins can set **budget controls** at organisation, cost centre, or individual level
- If the pool runs out: code completions still work; chat and agent mode pause until the next billing cycle or until more credits are purchased

---

2.4.5 Tips to stay within budget

- Use **Plan mode** before Agent mode - planning is cheaper than doing
- Pick the **lightest model** that can handle the task; save the most capable models for hard problems. The Auto mode will automatically attempt to pick the cheapest model for the task.
- Use skill libraries like Brave Copilot
- Keep context focused - avoid attaching entire codebases when a single file will do
- Monitor your usage via the GitHub Billing Overview page

2.4.6 Source

- GitHub blog: GitHub Copilot is moving to usage-based billing
- GitHub community discussion #192948

2.5 GitHub Copilot Ambassadors

Ambassadors are engineers and specialists across AFRY who actively promote and support GitHub Copilot adoption within their teams and disciplines.Let us know if you want to be added to the list.

Name	Email	Location	Area of Expertise
Anantharaman Velraj	anantharaman.velraj@afry.com	SE - Stockholm	Structural engineering, BIM/Revit plugin development
Antonio Malaguti	antonio.malaguti@afry.com	CH - Airolo	Civil site supervision, documentation automation
Anthony Vivek	anthony.jayanathvivek@afry.com	SE - Göteborg	AI/ML, CFD, fluid mechanics simulation
Björn Solem	bjorn.t.solem@afry.com	SE - Sundsvall	MicroStation/OpenRoads automation, road/rail BIM
Dragan Neskovski	dragan.neskovski@afry.com	SE - Malmö	BIM coordination, AutoCAD/Revit, quality assurance
Federico Albonico	federico.albonico@afry.com	CH - Zürich	BIM/Revit, water & environment infrastructure
Harri Orava	harri.orava@afry.com	SE - Lund	VBA, industrial automation, Citect/Plant SCADA
Henrik Landström	henrik.landstrom@afry.com	SE - Karlstad	AVEVA E3D/PML scripting, plant & process design
Johan Blomgren	johan.blomgren@afry.com	SE - Gävle	Civil 3D, water & road design
Johannes Götz	johannes.goetz@afry.com	DE - Mannheim	ArcGIS Pro, FME, environmental planning
Marcus Gripenberg	marcus.gripenberg@afry.com	SE - Göteborg	Civil 3D/Novapoint/Quadri, BIM data coordination
Marcus Öhman	Marcus.Ohman@afry.com	SE - Jönköping	Revit API, AutoCAD .NET, C#, electrical BIM
Matheus Rangel Venturi	matheus.v.venturi@afry.com	BR - São Paulo	AVEVA E3D, Caesar II, Revit, piping design
Mert Köylüoglu	mert.koyluoglu@afry.com	SE - Stockholm	BIM coordination, MagiCAD, sprinkler design
Oliver Miilus-Larsen	oliver.miilus-larsen@afry.com	SE - Malmö	GIS (ArcGIS Pro, QGIS, FME), urban development
Pedram Tahmoury	pedram.tahmoury@afry.com	SE - Göteborg	BIM management, Trimble Connect extensions
Rando Matti	rando.matti@afry.com	SE - Stockholm	Revit API C#, Tekla Open API, structural BIM automation
Sanjay Joshi	sanjay.joshi@afry.com	FI - Vantaa	Parametric/computational design, bridges, Dynamo/Python, FEM
Sergey Semenov	sergey.semenov@afry.com	SE - Örebro	Structural engineering, Grasshopper scripting
Siddarth K	siddarth.k@afry.com	IN - Noida	Rail/road design, Civil 3D/Bentley, bulk automation

3. Getting Started

3.1 Installation instructions

3.1.1 Installations

To be able to use GitHub Copilot efficiently, you will need a code editor and some programming languages. You will also benefit from having Git, which is a version handling tool. Because it sometimes takes a little while to get approval for installations.

We are aware that there are a lot of installations required. We are working on setting up an installation package that covers everything in the Application Kiosk.

In the bottom of this page is a checklist to ensure that everything is working.

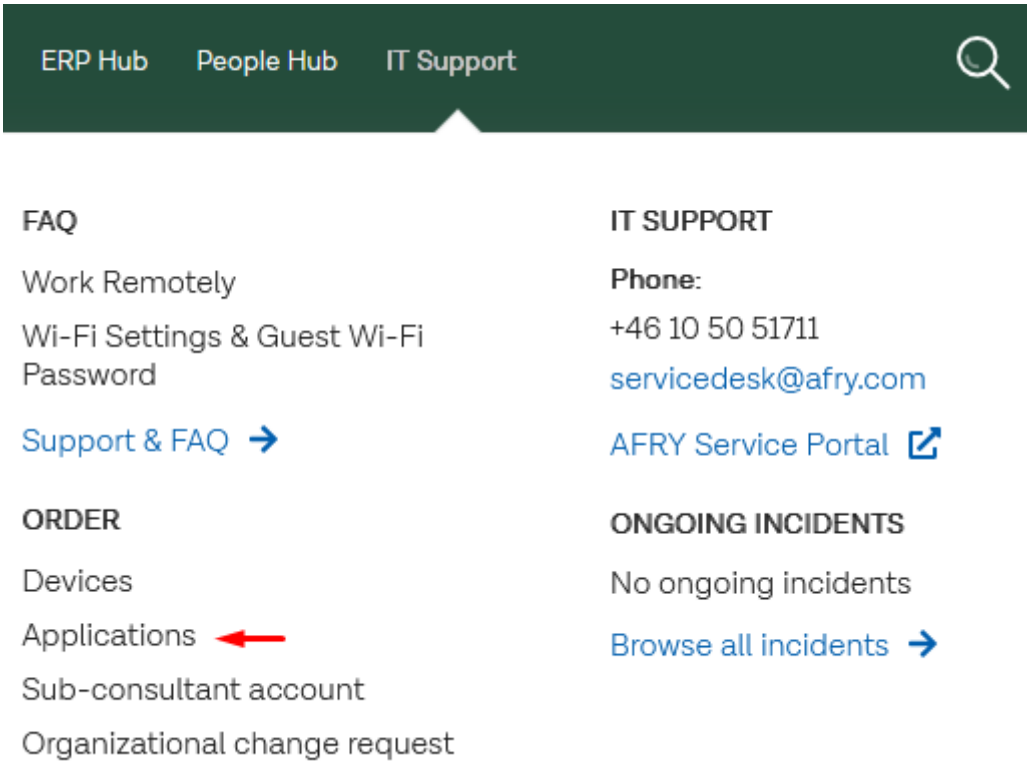
What to Install - Quick Overview

Priority	Tool	Purpose
🔴**Essential**	**VS Code**	Your code editor - everything runs here
🔴**Essential**	**GitHub Copilot** (via Application Kiosk)	The AI assistant - request approval ASAP as it needs manager sign-off
🟡 **Recommended**	**Git**	Version control - save and restore your work
🟡 **Recommended**	**Python**	Most widely used scripting language; also required by many MCP tools
🟡 **Recommended**	**Node.js**	Required for JavaScript-based tools and many MCP servers
⚪**Optional**	**GitHub CLI (`gh`)**	Only needed if you plan to publish websites to GitHub
⚪**Optional**	**Azure CLI (`az`)**	Only needed if you plan to deploy to Azure

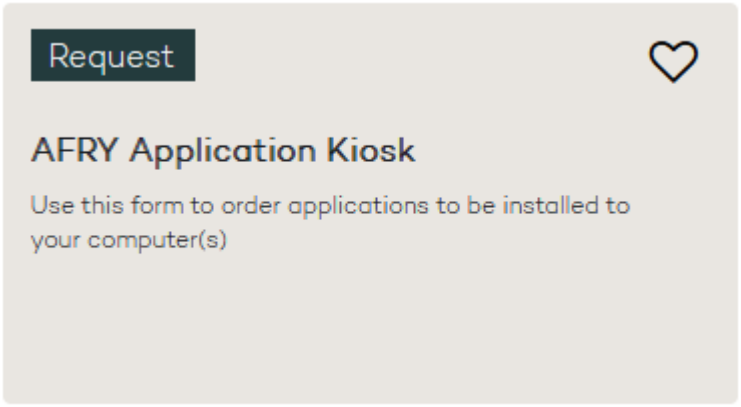
3.1.2 Installing GitHub Copilot and VS Code

GitHub Copilot is available as an extension for Visual Studio Code. VS Code can be downloaded for free from the internet, while GitHub Copilot requires a licence that must be requested through the Application Kiosk.

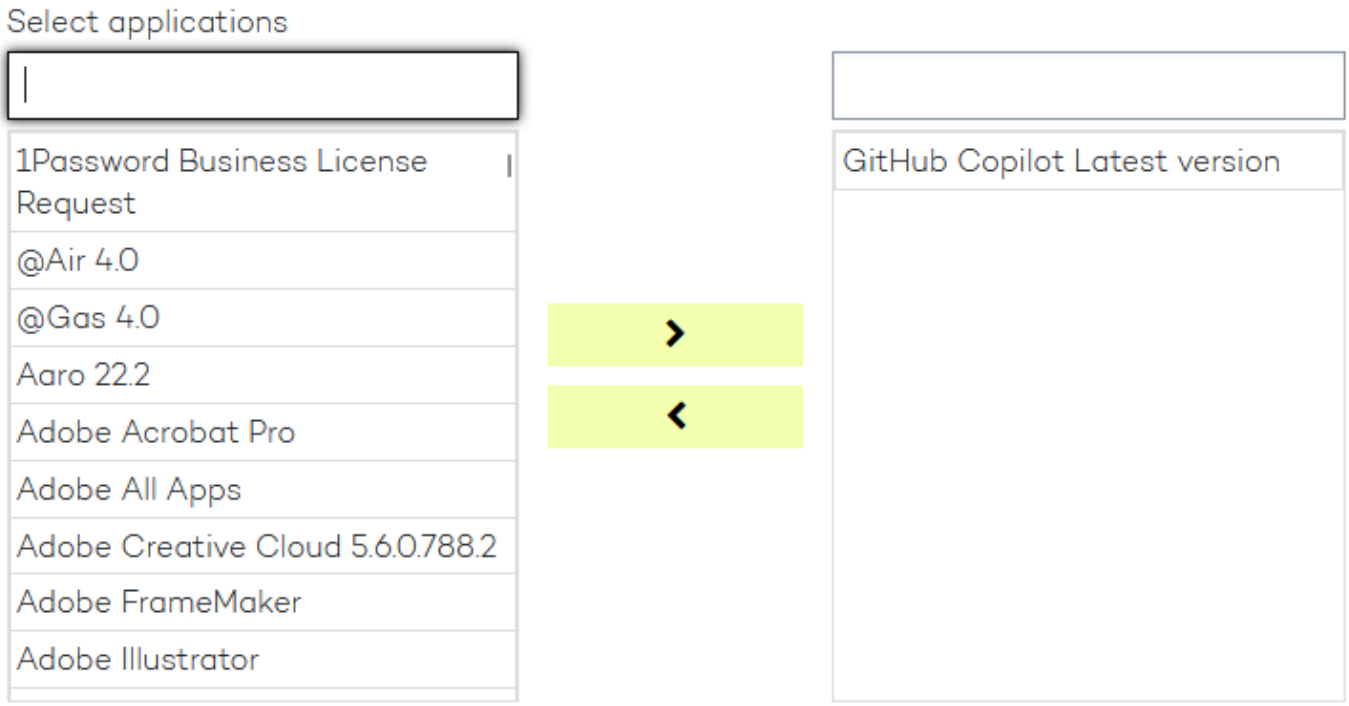
1. **Download and install VS Code**:
2. Go to https://code.visualstudio.com/
3. Click the **Download for Windows** button to download the installer
4. Run the installer and follow the default settings
5. It is recommended to check **"Add to PATH"** during installation so you can open VS Code from the terminal
6. **Admin rights**: VS Code requires admin rights to install - launching the installer should trigger a ServiceNow request automatically
7. **Request GitHub Copilot** from the AFRY Application Kiosk:
8. Visit the application portal page



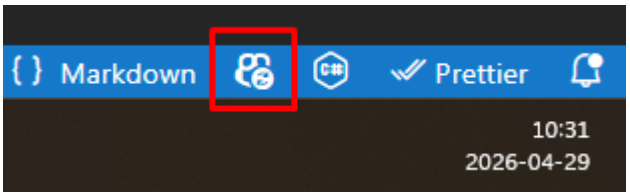
9. Click on the AFRY application kiosk box



10. Search and select **GitHub Copilot**



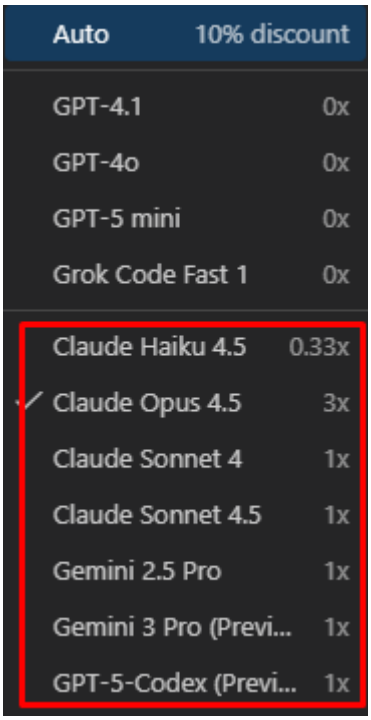
11. GitHub Copilot needs manager approval - wait for approval before continuing
12. **Sign into GitHub Copilot**:
13. Open Visual Studio Code
14. Click the small white Github Copilot icon in the bottom right corner.



15. Click **Use AI features**
16. Your browser will open for authentication
17. Your GitHub Enterprise username is in the email you received when registering (the user name is derived from your AFRY email)
18. When you enter the user name the password field will be disabled (ensure there is no whitespace at the end of the username). In the next screen you are taken to AFRY's login page.
19. **Validate GitHub Enterprise Account**:
20. Click the small chat icon at the very top of VS Code to open Copilot Chat

21. Type a simple question (e.g., "How do I create a Python function?") to verify the chat is working
22. Check your account status: Click the **GitHub Copilot** icon in the bottom status bar

23. It should show as **Active** or **Enabled**
24. **Check available AI models**:
25. In the Copilot Chat window, click on the **Auto** drop down.
26. Ensure you have access to premium models (Claude Sonnet 4.6, GPT-5.4, etc.)



27. If you only see free/limited models, your account may not be fully activated

---

3.1.3 Git

Git is a version handling program that allows you to collaborate with others, but also to manage your own work. With Git you can

- Store the code and resources for your projects centrally (in Github for example), so that you and others can access them from any computer
- Work on your own computer with versions - giving you restore points when things spiral out of control

Do I have it?

To check, you need to open a terminal. Press `Win + X` and select **Windows Terminal** or **PowerShell**. Alternatively, press `Win + R`, type `powershell`, and press Enter.

In the terminal window, type the following command and press Enter:

```
git --version
```

If you see a version number (e.g., `git version 2.43.0.windows.1`), Git is already installed. If the command is not recognized, you need to install it.

How do I install it?

Download the Windows installer from https://git-scm.com/install/windows

You need admin rights to install Git, but launching the installer should trigger ServiceNow to give you permission. Use the default settings during installation.

Did the installation work?

After installation, close and reopen your terminal (important - the old terminal won't see the new installation). Then run:

```
git --version
```

You should see the version number. You can also configure your identity (used for commits):

```
git config --global user.name "Your Name" git config --global user.email "your.email@afry.com"
```

3.1.4 Python

Python is a scripting programming language which is very powerful for data processing, although there are libraries for doing almost anything. Many MCP tools are created with Python, so you should install it even if you have other language preferences.

Do I have it?

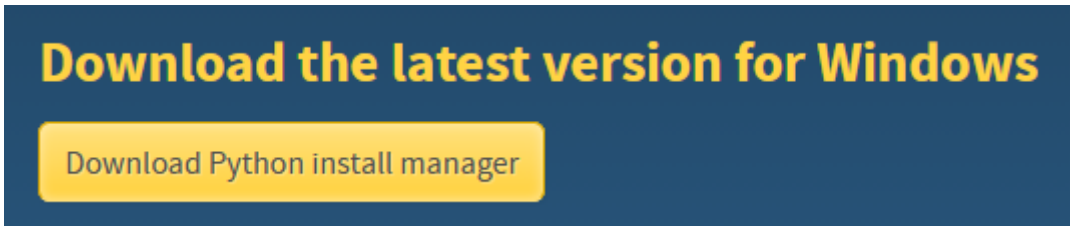
Open a terminal (press `Win + X` and select **Windows Terminal** or **PowerShell**) and run:

```
python --version
```

If you see a version number (e.g., `Python 3.11.5`), Python is already installed. If the command is not recognized or opens the Microsoft Store, you need to install it.

How do I install it?

Download an installer from https://www.python.org/downloads/ by clicking



Important: During installation, check the box **"Add Python to PATH"** on the first screen.

Note: If you plan on working with neural networks, to fine-tune language models or do other type of work with PyTorch/TensorFlow, you should choose Python v3.10 or v3.11.

Did the installation work?

After installation, close and reopen your terminal, then run:

```
python --version pip --version
```

Both commands should return version numbers. You can also test Python by running:

```
python -c "print('Hello from Python!')"
```

3.1.5 NodeJS

Node.js is a JavaScript runtime that allows you to run JavaScript outside of a web browser. Even if you do not plan to create websites, having NodeJS is still useful. Many tools (MCP) are distributed as packages which require NodeJS.

Do I have it?

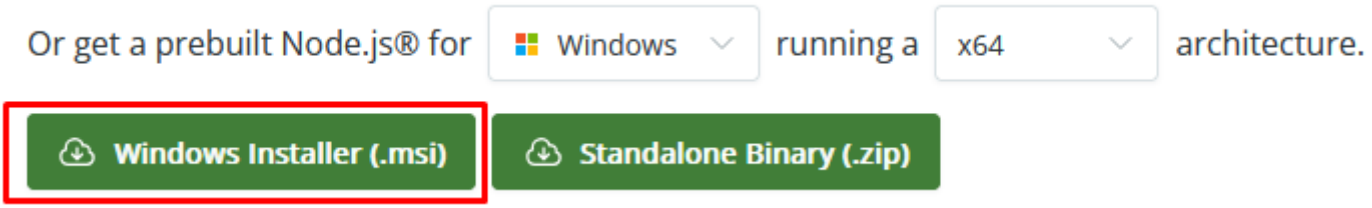
Open a terminal (press `Win + X` and select **Windows Terminal** or **PowerShell**) and run:

```
node --version npm --version
```

If you see version numbers for both (e.g., `v20.10.0` and `10.2.3`), Node.js is already installed. If the commands are not recognized, you need to install it.

How do I install it?

Download the Windows Installer msi from https://nodejs.org/en/download



Use the default settings during installation. The installer includes npm (Node Package Manager).

Did the installation work?

After installation, close and reopen your terminal, then run:

```
node --version npm --version
```

Both commands should return version numbers. You can also test Node.js by running:

```
node -e "console.log('Hello from Node.js!')"
```

3.1.6 GitHub CLI (GH CLI)

Note: GH CLI is only needed if you plan to develop and share **websites**. If you are writing plugins for software or doing other local development, you can skip this.

GitHub CLI (`gh`) lets you interact with GitHub directly from the terminal - creating repositories, managing pull requests, triggering deployments, and more.

Do I have it?

Open a terminal and run:

```
gh --version
```

If you see a version number (e.g., `gh version 2.40.0`), it is already installed.

How do I install it?

Download the Windows installer from https://cli.github.com/

Run the installer with default settings. Admin rights may be required.

Did the installation work?

After installation, close and reopen your terminal, then run:

```
gh --version
```

You can also authenticate with your GitHub account:

```
gh auth login
```

Follow the prompts and select **GitHub Enterprise Server** if using AFRY's GitHub.

3.1.7 Azure CLI (AZ CLI)

Note: AZ CLI is only needed if you plan to **deploy websites or other resources to Azure**. If you are writing plugins for software or doing other local development, you can skip this.

Azure CLI (`az`) allows you to manage Azure resources from the command line - deploying web apps, configuring services, and managing cloud infrastructure.

Do I have it?

Open a terminal and run:

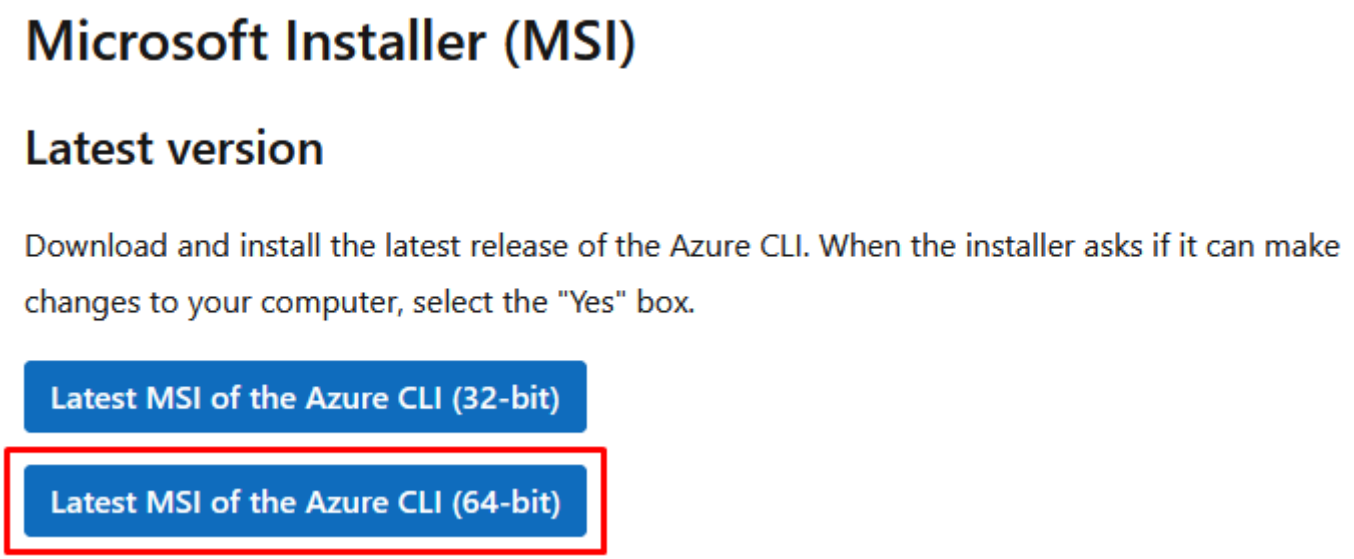
```
az --version
```

If you see a list of version numbers, it is already installed.

How do I install it?

Download the MSI installer from https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-windows?view=azure-cli-latest&pivots=msi

The button to download the installer is found further down on the page. Run the installer with default settings. Admin rights may be required.



Did the installation work?

After installation, close and reopen your terminal, then run:

```
az --version
```

You can also log in to Azure:

```
az login
```

This will open a browser window for authentication using your AFRY account.

3.1.8 Installation Checklist

Use this checklist to verify all software has been installed correctly.

Software	How to Check	Expected Result	✓
VS Code	Open terminal: `code --version`	Version number (e.g., `1.85.0`)	☐
GitHub Copilot	Open VS Code → Extensions → Search "GitHub Copilot"	Shows "Installed"	☐
GitHub Copilot Status	Click GitHub Copilot icon in bottom status bar	Shows "Active" or "Enabled"	☐
GitHub Copilot Chat	Click chat icon at top or press Ctrl+Alt+I	Chat window opens and responds to questions	☐
AI Model Access	Open Copilot Chat → Check model dropdown at bottom	Multiple models available (Claude Sonnet 4.5, GPT-4o, etc.)	☐
Git	`git --version`	Version number (e.g., `git version 2.43.0`)	☐
Python	`python --version`	Version number (e.g., `Python 3.11.5`)	☐
pip	`pip --version`	Version number with Python path	☐
Node.js	`node --version`	Version number (e.g., `v20.10.0`)	☐
npm	`npm --version`	Version number (e.g., `10.2.3`)	☐
GH CLI *(optional)*	`gh --version`	Version number (e.g., `gh version 2.40.0`) - needed for website deployment	☐
AZ CLI *(optional)*	`az --version`	List of version numbers - needed for Azure deployment	☐

3.1.9 Troubleshooting

If any command is not recognized:

1. **Close and reopen your terminal** - New installations won't be visible in terminals that were already open
2. **Check PATH** - The program may be installed but not added to your system PATH
3. **Restart your computer** - Some installations require a restart to take effect
4. **Reinstall** - If issues persist, try reinstalling the software

3.2 Getting Started with VS Code and GitHub Copilot

This tutorial walks you through opening VS Code, finding the GitHub Copilot chat window, and running your first AI-assisted coding session - from data creation to a working web app.

📹 **Prefer to watch?** See the video walkthrough.

3.2.1 Prerequisites

- VS Code installed (download here)
- GitHub Copilot extension installed and signed in
- Python installed on your machine (for the example app)

3.2.2 1. Opening a Folder in VS Code

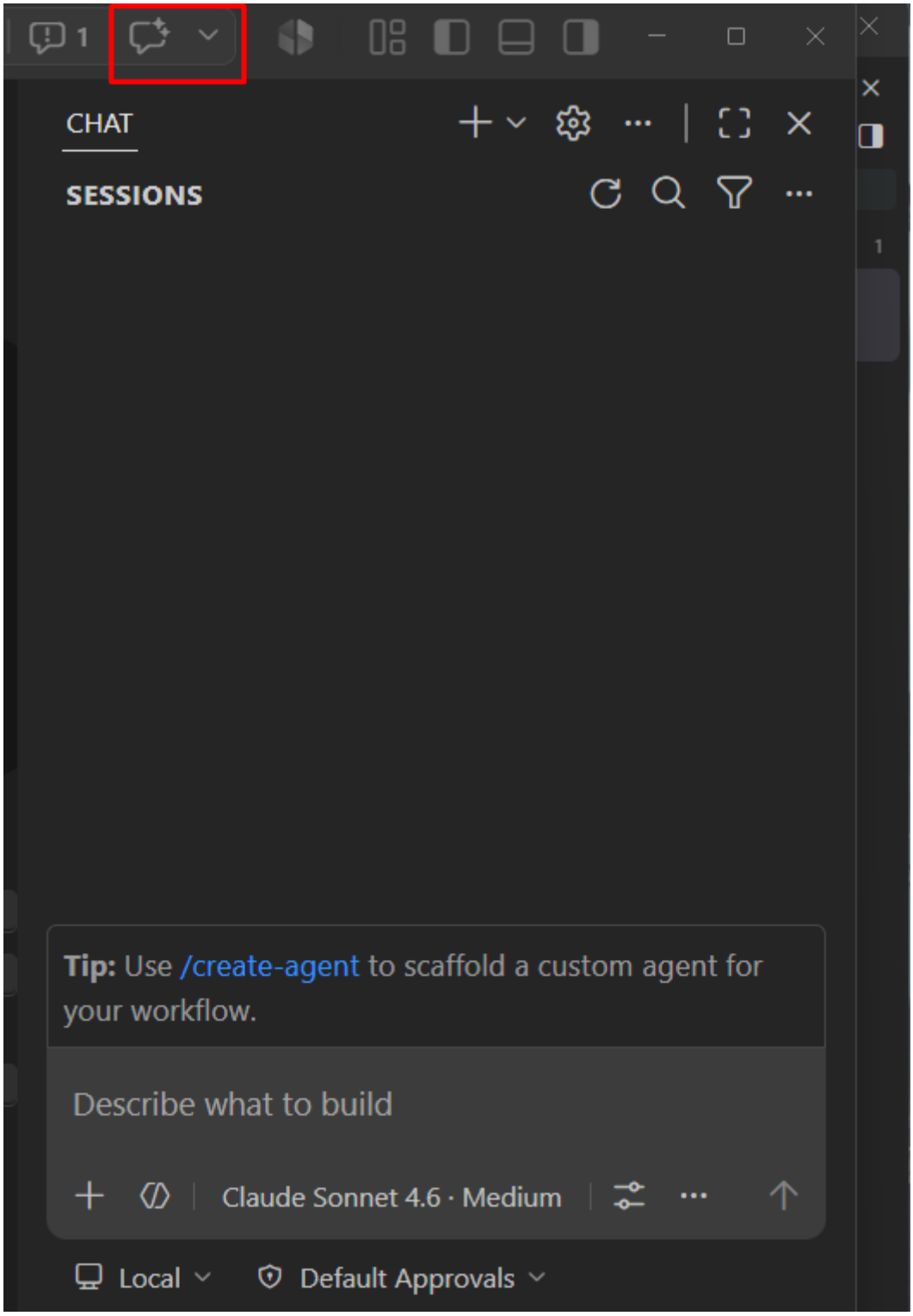
Before you start, open the folder where you want to work.

1. Launch VS Code.
2. Go to **File → Open Folder...** (or press `Ctrl+K Ctrl+O`).
3. Select a folder on your computer - for example, a new empty folder called `my-first-copilot-project`.
4. Click **Select Folder**.

VS Code will now show the folder contents in the **Explorer** panel on the left.

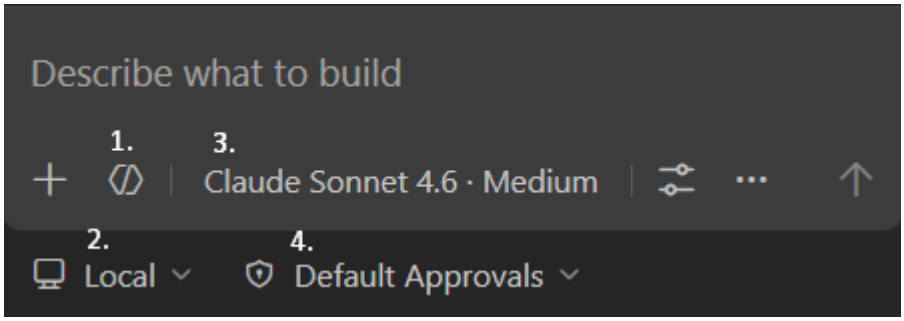
Tip: Working inside an open folder gives Copilot context about your project - it can read and write files, run commands, and understand your code structure.

3.2.3 2. Opening the GitHub Copilot Chat Window



1. If the window is not open, you can click the **Chat icon** in the top bar (it looks like a speech bubble).
2. The Copilot Chat panel will open on the side of your editor.

3.2.4 3. Selecting an Agent



GitHub Copilot Chat is built around **agents** - each with a specific role. For this tutorial, use the **Agent** agent, which can take actions on your behalf: creating files, running commands, and building projects end to end.

1. At the top of the chat panel, check the **Agent Target** dropdown and make sure **Local** [2] is selected. This means the agent runs interactively on your own machine with full access to your workspace.
2. In the chat input box, click the **agents dropdown** [1] (it may show **Ask** by default).
3. Select **Agent** from the list.

Agent	Description
Agent	Autonomously plans and implements changes across files, runs terminal commands, and invokes tools
Plan	Creates a structured, step-by-step implementation plan before writing any code - hands it off to Agent when you're happy with it
Ask	Answers questions about your code or general topics without making file changes

---

3.2.5 4. Selecting a Model

You can choose which AI model powers your Copilot session. Different models have different strengths.

1. In the chat panel, click the **model selector** [3] (shown near the chat input box).
2. Select **Claude Sonnet 4.6** (or whichever model you prefer).

Tip: Claude Sonnet 4.6 is a strong general-purpose model well-suited for coding tasks and multi-step agent workflows.

3.2.6 4b. Choosing a Permission Level

When the **Agent** agent is active, it will ask your permission before running terminal commands or making changes. The **permissions picker** [4] lets you control how much autonomy it has:

Level	Behaviour
Default Approvals	Only read-only / safe tools run without confirmation - everything else prompts you
Bypass Approvals	All tool calls are auto-approved; the agent may still ask clarifying questions
Autopilot	Fully autonomous - auto-approves everything and continues until the task is done

For this tutorial, leave it on **Default Approvals** so you can see what the agent asks before it acts.

3.2.7 5. Your First Task: Create a Data File

Now let's put Copilot to work. With the **Agent** agent and **Claude Sonnet 4.6** selected, type the following into the chat:

Please create a CSV file which contains the largest cities in the world, including their populations and coordinates

Copilot will:

- Generate a `cities.csv` file with city names, populations, latitudes, and longitudes.
- Write the file directly into your open folder.

You can review the file in the Explorer panel once it's created.

3.2.8 6. Your Second Task: Build a Web App

Now ask Copilot to build a visualization on top of that data:

Please create a Python web app which allows me to visualize and edit this data

Copilot will likely:

1. Create a Python web application (e.g., using Flask or Streamlit).
2. Set up a map or table view to display the city data.
3. Add editing functionality so you can update the CSV entries.
4. Tell you how to start the app - or you can ask it to start the app for you.

Letting Copilot Start the App

If Copilot gives you manual start instructions, you can simply ask:

Can you start the app for me?

Copilot will run the necessary terminal commands to launch the app and give you a local URL to open in your browser (typically `http://localhost:5000` or similar).

3.2.9 7. Understanding Permission Prompts

When running in Agent mode, Copilot will sometimes **ask for your permission** before taking an action - such as running a terminal command, installing a package, or modifying a file.

Do not blindly accept these prompts. Take a moment to read what Copilot is asking to do. If you are unsure, ask:

What does this command do and why do you need to run it?

Copilot will explain its reasoning. This is a healthy habit that keeps you in control of your own environment.

3.2.10 Summary

You have now:

- Opened a folder in VS Code
- Found and configured the Copilot Chat window
- Selected the **Agent** agent with the **Local** target
- Used Copilot to autonomously create a data file and a working web app

---

3.2.11 Next Steps

- Use the **Plan** agent to design a feature before building it - review the plan, then hand it off to **Agent** to implement
- Learn about prompting techniques to get better results
- Try adding a new feature to your cities app - just describe it to Copilot!

3.3 Spec-Driven Development with GitHub Copilot

You don't need to know what a specification is, or how to write one, to work in a structured, deliberate way with Copilot. The **Plan** agent does that for you - it interviews you about what you want, asks the right clarifying questions, and turns your answers into a concrete implementation plan before a single line of code is written.

This section is designed as a tutorial which walks through the full cycle using a real example: a Python tool that reads an Excel timesheet export from Maconomy and produces a project hours summary.

A sample file is included in this folder: **Fictional Timesheet.xlsx** - a realistic Maconomy export covering two years of consultant time, with three sheets (Week, Month, Year), ~1,300 data rows, and columns like `Client Working Hours`, `Non-Client Working Hours`, `Absence`, and ISO `Week` numbers. Use it as your test data throughout the exercise.

3.3.1 Why This Workflow?

Most people jump straight to asking Copilot to write code. That works for small tasks - but for anything beyond a few lines, it leads to code that almost-but-not-quite does what you wanted, which you then spend time patching rather than thinking clearly. The workflow in this tutorial builds good habits:

Phase	What you do	Why it matters
Plan	Describe what you want; Copilot asks questions and writes a spec	Catches misunderstandings*before* they become code. Cheap to change a plan; expensive to change working code.
Implement	Copilot writes the code step by step, you review each change	You stay in control. Every change is shown as a diff - read it and push back if it's wrong.
Run it	Copilot runs the tool with your test data	Confirms the code actually works end-to-end before you invest time in tests.
Test	Copilot writes automated tests for the behaviour you described	Tests prove the code does what you agreed. They catch regressions when you change things later. You don't need to know how to write tests - just ask.
Code review	Copilot reviews what it wrote, looking for bugs and gaps	AI-generated code contains mistakes. A second pass - especially with a*different* model - catches things the first pass missed.

3.3.2 Before You Start

Open VS Code in an **empty folder** - this will be your project. All files Copilot creates will land here.

Create the folder anywhere you like, then open it: **File → Open Folder**.

Copy **Fictional Timesheet.xlsx** from this tutorial folder into your project folder. You'll use it as test data throughout the exercise.

3.3.3 The Idea: Project Hours Summary Tool

AFRY consultants often export their logged hours from Maconomy into Excel. The resulting file is more complex than a flat table - it has three sheets at different levels of granularity (daily, monthly, yearly), separate columns for chargeable and non-chargeable hours, and ISO week numbers. Summarising that manually - utilisation rates, totals per project, trends by week - is tedious and error-prone.

We'll build a small Python tool that:

- Reads the **Week** sheet from a Maconomy `.xlsx` export
- Calculates **chargeable utilisation** - client hours as a percentage of total working hours
- Summarises hours **per project** and **per ISO week**
- Optionally filters by date range or project code
- Prints the summary to the terminal

You don't need to know any of this in advance - you'll just describe the rough idea to Copilot, and the Plan agent will figure out the details with you.

3.3.4 Step 1: Inspect the Data

Before starting the planning interview, give Copilot a chance to look at the actual file. This makes the planning conversation much sharper - instead of asking generic questions about column names, Copilot will ask specific, useful questions based on what it can see.

Use the regular Agent for this step, not Plan. Plan mode cannot run terminal commands, so it cannot open or inspect an Excel file. You'll switch to Plan in the next step.

1. Open the Chat panel (`Ctrl+Alt+I`)
2. Click the **paperclip icon** in the chat input bar and attach **Fictional Timesheet.xlsx** (or drag it directly into the chat)
3. Ask Copilot to describe what it sees:

```
I've attached a Maconomy timesheet export. Can you describe its structure? How many sheets does it have, what are the column name
```

Copilot will enumerate the three sheets, the column names, and what the data looks like.

Why this matters: Without seeing the file, Copilot asks generic questions and makes assumptions. With the file attached, it asks sharper questions - "*should I use the Week sheet or the Month sheet?*" and "*should absence days be excluded from the utilisation calculation?*" - which lead to a much more accurate implementation plan.

3.3.5 Step 2: Create the Plan

Open the Chat panel (`Ctrl+Alt+I`) and select **Plan** from the agents dropdown.

The file context from Step 1 is still available in the conversation - no need to attach it again. Describe your idea:

```
I want to build a Python tool that reads this Maconomy timesheet export and gives me a utilisation summary - how much of my time
```

That's it. Press Enter.

The Interview

The Plan agent will respond with clarifying questions. This is intentional - it's extracting the requirements from you rather than making assumptions. Answer honestly and conversationally.

A typical exchange might look like:

Copilot: The file has three sheets - Week, Month, and Year. Which should I use as the primary data source?

You: The Week sheet. It has one row per project entry per day, which gives the most flexibility for filtering and grouping.

Copilot: The Week sheet has both `Client Working Hours` and `Non-Client Working Hours` columns. Should utilisation be based only on client hours, or should I include non-client hours in the total too?

You: Total hours in the denominator (client + non-client), client hours in the numerator.

Copilot: Should absence days be excluded from the utilisation calculation, or treated as non-chargeable time?

You: Excluded. I only want to measure utilisation across days I was actually working.

Copilot: Should the project summary show `Subproject No.` (e.g. `GBR002100-001`), `Subproject Name`, or both?

You: Both - the code and the name next to it.

Copilot: Should this be a command-line tool with arguments, or a script you just run directly?

You: CLI. At minimum an argument for the input file path, and optionally a date range.

Keep answering until Copilot stops asking. If you're unsure about something, say so - "I don't know, you decide" is a perfectly valid answer.

3.3.6 Step 3: Review and Refine the Plan

Once the interview is done, the Plan agent produces a structured implementation plan. It will look something like:

```
## Implementation Plan: Maconomy Hours Summary Tool 1. Set up project structure - Create hours_summary.py - Add requirements.txt
```

Read the plan carefully. This is the moment to catch misunderstandings before they become code.

- Does it match what you had in mind?
- Is there anything missing or wrong?

Push back with follow-up messages if needed:

"*I don't want pandas as a dependency - use openpyxl directly*"

"*Add a --week argument so I can filter by ISO week number*"

"*Don't bother with the output Excel file for now - just terminal output*"

Iterate until the plan is right. **It's cheap to change a plan. It's expensive to change code.**

When you're happy, click **Open in editor** - this opens the plan as an editable document in VS Code. Read through it once more and make any final tweaks directly in the file.

Tip: The Plan agent automatically saves the plan to `/memories/session/plan.md`. You can retrieve it later via the Command Palette: `Chat: Show Memory Files`.

Once you're done reviewing, open a **new chat** (`Ctrl+Alt+I`), make sure the regular **Agent** is selected, and kick off the implementation:

```
Please implement the plan in #file:plan.md
```

3.3.7 Step 4: Implement

After handing off from Plan, the **Agent** agent takes over. It will:

- Create the project files and folder structure (`hours_summary.py`, `requirements.txt`, etc.)
- Write the implementation following the plan step by step
- Run terminal commands as needed (e.g., `pip install openpyxl tabulate`)
- Show you inline diffs for every file change

What to Watch For

Review each file change before accepting. The agent shows diffs inline - use the overlay controls to **Keep** or **Undo** individual changes. Don't accept everything blindly.

Answer permission prompts honestly. If the agent asks to run a terminal command you don't recognise, ask it first:

What does this command do and why is it needed?

Steer mid-flight if needed. If the implementation diverges from what you discussed in the plan, say so:

The output should show a table grouped by project first, then employees within each project - not a flat list.

Keeping the Agent on Track

If the conversation grows long, the agent may start to drift. You can re-anchor it to the plan at any point:

```
Look at the plan in #file:plan.md. The --project filter hasn't been implemented yet. Please implement that next.
```

3.3.8 Step 5: Run It

Before writing tests, confirm the tool actually works end-to-end with your sample data.

Ask the Agent:

```
Please run the tool using Fictional Timesheet.xlsx as the input file.
```

The Agent will run the command in the terminal and show you the output. Check:

- Does it print output without errors?
- Do the numbers look plausible? (Spot-check a week manually against the raw data.)

If something is wrong, describe what you expected and let the Agent fix it before moving on to tests.

3.3.9 Step 6: Write Tests

You don't need to know how to write tests, or even what to test - just ask:

```
Please write a thorough set of unit tests for this tool.
```

The Agent will read the implementation and generate a test file covering the main behaviours. It will also run the tests and fix any failures it finds.

Once the initial tests are passing, push for more coverage:

- **Edge cases**: `"Add a test for a file where all hours are 0"`
- **Invalid inputs**: `"Test what happens if the Hours column contains text instead of numbers"`
- **Boundary conditions**: `"Test with a single row, and with an empty sheet"`
- **Your specific requirements**:

  `"Add a test that confirms absence rows are excluded from the utilisation denominator"`

Test-First Variant

If you want to try a more disciplined approach, you can ask for tests *before* the implementation. Go back to Step 3, and before opening the plan in the editor, add:

```
Before we implement, please also add a set of failing unit tests to the plan that verify the key behaviours. The implementation s
```

3.3.10 Step 7: Code Review

AI-generated code is not always correct - it can contain bugs, miss edge cases, and make poor design decisions. Asking Copilot to review what it just wrote is a cheap way to catch problems before they cause trouble.

Tip: switch models before reviewing. The model that wrote the code is likely to miss the same things it missed when writing. Switch to a different model in the chat model picker - a fresh perspective catches more.

Self-Review Before Committing

```
Review #file:hours_summary.py for: 1. Potential bugs or unhandled edge cases 2. Input validation - what if the Excel file has une
```

Spec Compliance Check

Reference the plan Copilot generated earlier:

```
Based on our earlier implementation plan, does hours_summary.py implement everything we agreed on? Are there any gaps?
```

Focused Review Prompts

Robustness:

```
What happens if the Hours column contains a blank cell or the text "n/a"? Is this handled gracefully?
```

Correctness:

```
Does the per-project summary correctly aggregate hours when the same employee appears across multiple rows for the same project?
```

What Copilot Catches Well vs. What Needs Human Eyes

Copilot catches well	Needs human judgement
Missing error handling	Whether the output format is actually useful
Off-by-one and aggregation errors	Whether the column names match your real files
Unhandled file/IO exceptions	Whether the filtering logic fits your team's workflow
Inconsistency with the agreed plan	Performance with very large files

Implementing the Fixes

Once the review is done, apply everything in one go:

```
Please apply all the fixes you identified.
```

3.3.11 Full Workflow Summary

```
[Agent] Inspect the data file ↓ [Plan] Describe your idea → answer interview questions ↓ Review and edit the plan (Open in editor
```

3.3.12 Tips for Getting the Best Results

You Don't Need to Know Everything Upfront

The whole point of starting in Plan mode is that Copilot extracts the requirements from you through conversation. If you don't know the answer to one of its questions, say so - it will suggest a reasonable default.

Iterate on the Plan Before Implementing

It's cheap to change a plan. It's expensive to change code. Spend time in Plan until you're genuinely happy with the steps before clicking Start Implementation.

Re-anchor With the Plan if the Agent Drifts

In a long session, the Agent can forget earlier decisions. Remind it:

```
Refer back to the plan we agreed on and check you haven't missed anything.
```

Review Diffs, Don't Blindly Accept

Every change the Agent makes is shown as a diff. Read them. You are responsible for the code - Copilot is your assistant, not your replacement.

Save the Plan as a Spec

After a successful session, ask Copilot to write the plan out as a `docs/spec.md` file. This becomes your documentation, your onboarding guide, and your starting point for the next session.

```
Please write a clean specification document based on everything we built and save it as docs/spec.md
```

3.3.13 Extension Exercises

Once the core tool works, try these - each one requires a new planning conversation.

1. Utilisation trend chart

```
Add a matplotlib bar chart showing my monthly utilisation % over the full dataset. Each bar = one month, coloured green if above
```

2. Busiest and quietest weeks

```
Extend the tool to print my 5 highest- and 5 lowest-utilisation weeks, with the date range of each week and the top project worke
```

3. Project timeline

```
Add a --timeline flag that prints a text-based Gantt chart showing which projects were active in which months, based on the first
```

4. Test-first variant

Go back to Step 1 and this time, before clicking Start Implementation, add:

```
Before implementing, write failing unit tests that cover: - Correct utilisation calculation when absence rows are present - Corre
```

Then hand off to Agent and watch it make the tests pass.

3.3.14 Next Steps

- Try the same workflow on a real task from your own work
- After the session, ask Copilot to generate the spec document from the plan - this is how spec-driven development starts to happen naturally
- Explore custom agents to encode your team's preferred planning style into a reusable agent

3.4 System Instructions and README

Copilot doesn't know anything about your project unless you tell it. By default, every conversation starts from scratch - no conventions, no preferences, no awareness of how your team works or what libraries you use.

Two files change that: `copilot-instructions.md` and `README.md`. Together they give Copilot the context it needs to behave consistently and produce output that fits your work from the very first message.

This applies whether you're a developer, an analyst writing Python scripts, or a project manager using Copilot to draft documents - the same mechanism works for all of them.

3.4.1 What Are System Instructions?

System instructions are plain text files that Copilot reads automatically at the start of every conversation. You don't need to mention them in your prompt - they're always active in the background.

The primary file is `.github/copilot-instructions.md` in the root of your repository. Copilot loads it silently and treats its contents as standing instructions for the session.

What to put in it

Think of it as a briefing document for a new colleague joining your project. Write down the things you'd want them to know before they start:

- **Coding conventions** - indentation, file naming, preferred patterns
- **Preferred libraries** - "use `axios` for HTTP, not `fetch`", "prefer `zod` for schema validation"
- **Things to avoid** - "never use `var`", "don't add inline comments unless asked"
- **Folder structure** - "scripts go in `scripts/`, reports go in `output/`"
- **Testing expectations** - "write xUnit tests, not NUnit; use `FluentAssertions`"
- **Language** - "all output in English", or "write in Swedish when the user writes in Swedish"
- **How Copilot should behave** - tone, verbosity, and honesty (more on this below)

A short, focused file is better than a long exhaustive one. Include what matters most; leave out what Copilot would figure out from the code or context anyway.

Example

```
# Copilot Instructions ## Language - All code, comments, commit messages, and variable names must be in English. ## C# conventio
```

3.4.2 Controlling Behaviour: Tone, Verbosity, and Sycophancy

System instructions aren't only for code conventions. You can also use them to shape how Copilot communicates - which turns out to matter more than most people expect.

Verbosity

By default, Copilot adds introductions, summaries, and explanations to most responses. That's fine when you're learning, but in a working session it creates noise. A single instruction cuts it significantly:

```
## Response style - Be concise. Skip introductions and closing summaries. - Don't explain what you just did - show the result. -
```

This also has a practical side effect: shorter responses use fewer tokens, which reduces cost if you're on a metered plan.

Sycophancy

Left unchecked, AI models tend to be agreeable - they validate ideas, soften criticism, and avoid disagreement even when disagreement would be more useful. This is called sycophancy, and it's a known limitation of large language models.

You can push back against it directly in your instructions:

```
## Honesty - Do not validate my ideas to be polite. If something is wrong, say so clearly. - If I ask you to review something, gi
```

Tone

If you use Copilot to draft documents, emails, or reports, you can set expectations about tone and style that persist across all your work:

```
## Writing style - Write in plain English. Avoid jargon and corporate language. - Use short sentences and short paragraphs. - Whe
```

These instructions apply to everything Copilot produces in the workspace - code, documents, emails, and chat responses alike.

3.4.3 Scoped Instructions

Not all rules apply everywhere. If your repository has a frontend and a backend, TypeScript conventions shouldn't apply to C# files.

Scoped instructions let you write `.instructions.md` files that only apply to specific folders or file types, using an `applyTo` pattern in the file's frontmatter:

```
--- applyTo: "src/frontend/**" --- ## TypeScript conventions - Use functional React components only - no class components. - Pref
```

Place the file anywhere in your workspace. When Copilot is working on a file that matches the `applyTo` pattern, the instructions are automatically included. When it's working elsewhere, they're ignored.

File type	When it applies
`.github/copilot-instructions.md`	Always - every conversation in the repo
`.instructions.md` with `applyTo`	Only when editing files matching the pattern
`.instructions.md` without `applyTo`	Always, like the root file

3.4.4 README.md as Context

A `README.md` is a plain text file placed in the root of your project folder. It's the front door to your project - the first thing anyone reads to understand what it does, how to set it up, and how to use it. GitHub displays it automatically when you open a repository in the browser.

A good README covers:

- **What the project does** - a short description of its purpose and who it's for
- **How to install and run it** - the exact commands needed to get started
- **Folder structure** - where things live and why
- **Key dependencies** - what the project relies on
- **Configuration** - environment variables, settings files, anything that needs to be set up before running

Copilot reads your README too. When you ask it to make a change, it scans your workspace for context - and a clear README gives it a significant head start:

- **What the project does** - helps Copilot understand intent and avoid off-topic suggestions
- **How to build and run it** - so it can write correct terminal commands
- **Folder structure** - so it knows where things belong when creating new files
- **Key dependencies** - so it doesn't suggest alternatives you've already decided against

A README that looks like this gives Copilot useful signal:

```
# Weekly Hours Summary Tool A Python command-line tool that reads a Maconomy timesheet export and prints a project hours summary
```

You don't need to write the README for Copilot's benefit specifically - a README that's useful for onboarding a new colleague will serve Copilot equally well.

3.4.5 Exercise: System Instructions in Action

The fastest way to understand how system instructions work is to see them change Copilot's behaviour in real time.

Step 1: Create the instructions file

In your project folder, create the file `.github/copilot-instructions.md` with the following content:

```
# Copilot Instructions ## Response style - Be concise. Skip introductions and closing summaries. - Do not use em dashes (-). Use
```

You can create the file manually in VS Code (right-click the Explorer panel and choose **New File**), or ask Agent to do it:

```
Please create .github/copilot-instructions.md with the content from the exercise.
```

Step 2: Start a new chat and test it

Open a **new chat** (`Ctrl+Alt+I`) - existing conversations don't pick up instruction file changes.

Then ask something simple, for example:

```
What is a README file and why does it matter?
```

Copilot's answer should be noticeably shorter than usual, avoid em dashes, and contain a Lord of the Rings reference somewhere.

Step 3: Reflect and adapt

Now you've seen instructions working, replace the LOTR rule with something real. Think about:

- A habit Copilot has that annoys you (e.g. long preambles, unsolicited refactoring)
- A convention from your team's code (e.g. naming patterns, preferred test framework)
- A tone preference for documents or emails you draft with Copilot

Update the file, start another new chat, and check whether the behaviour changed.

Tip: Instructions work best when they're specific and unambiguous. "Be professional" tells Copilot nothing it doesn't already know. "No filler phrases like 'Great question!' or 'Certainly!'" is something it can actually follow.

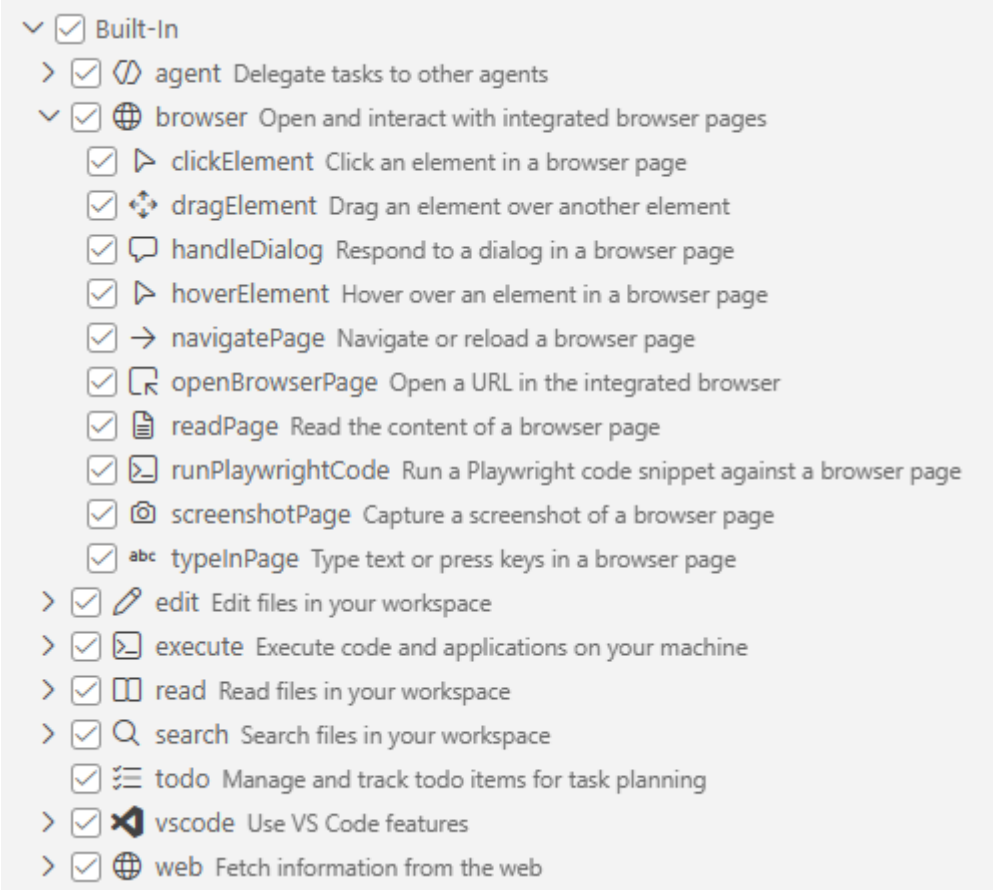
3.4.6 Summary

What	Controls
`copilot-instructions.md`	Conventions, structure, behaviour - always active for the whole repo
Scoped `.instructions.md`	Folder- or file-type-specific rules
`README.md`	Project context: purpose, structure, dependencies

The best instruction files are written incrementally - start small, and add a line every time Copilot does something you didn't want. After a few sessions, you'll have a file that reliably produces output that fits your project.

3.4.7 Introduction

When you chat with GitHub Copilot, it can do more than just generate text - it can **take actions**. It can read files, search the web, run terminal commands, and query databases. Each of these capabilities is a **tool**.



By default, Copilot comes with a set of built-in tools (file editing, terminal access, web search, etc.), but the real power comes when you connect **external tools** - and that's where **MCP** comes in.

3.4.8 What is MCP?

MCP (Model Context Protocol) is an open standard that lets AI assistants connect to external systems in a uniform way. Think of it as a USB-C port for AI: instead of building a custom integration for every service, MCP provides one protocol that works across all of them.

An MCP **server** is a small program that exposes tools to Copilot. For example:

- A **GitHub MCP server** lets Copilot create pull requests, search issues, and manage repositories
- A **database MCP server** lets Copilot query tables and inspect schemas
- A **Slack MCP server** lets Copilot send messages and search channels
- A **file transfer MCP server** lets Copilot move files between systems

When you connect an MCP server, its tools show up in Copilot's tool list. The AI then decides when to use them based on your prompt - you don't need to call them manually.

Why does this matter?

Without MCP, every AI assistant would need its own custom integration for every service - and those integrations wouldn't be reusable across different AI tools. With MCP:

- **One server works everywhere** - the same MCP server works in VS Code, Claude, Cursor, JetBrains, and 100+ other clients
- **You stay in your flow** - instead of switching between apps, you ask Copilot to do it
- **It's composable** - you can connect multiple MCP servers and Copilot will use them together in a single conversation

3.4.9 How it works

1. You **add an MCP server** to VS Code (either a local program or a remote URL)
2. Copilot **discovers the tools** the server provides
3. You **write a prompt** like you normally would
4. Copilot **picks the right tools** and asks for your approval before running them
5. The tools **execute and return results** that Copilot uses to continue the conversation

The rest of this guide walks through each step with hands-on examples.

3.4.10 AFRY MCP servers

There are several MCP servers available for AFRY related tasks. They allow you to generate images, build Powerpoint deck slides, find information about your organisation or to process files.

[List of servers]

Local and Remote MCP servers

MCP servers come in two flavors. **Local servers** run as a process on your machine. They typically require a programming environment such as NodeJS or Python. VS Code starts them automatically. These are great for tools that need access to your local files, Git repos, or development environment. **Remote servers** are hosted on the internet and accessed over HTTP. You connect to them with a URL, and they often require

authentication (e.g. OAuth). Remote servers are ideal for shared services like company APIs, cloud platforms, or SaaS integrations where the tool doesn't need to run on your laptop.

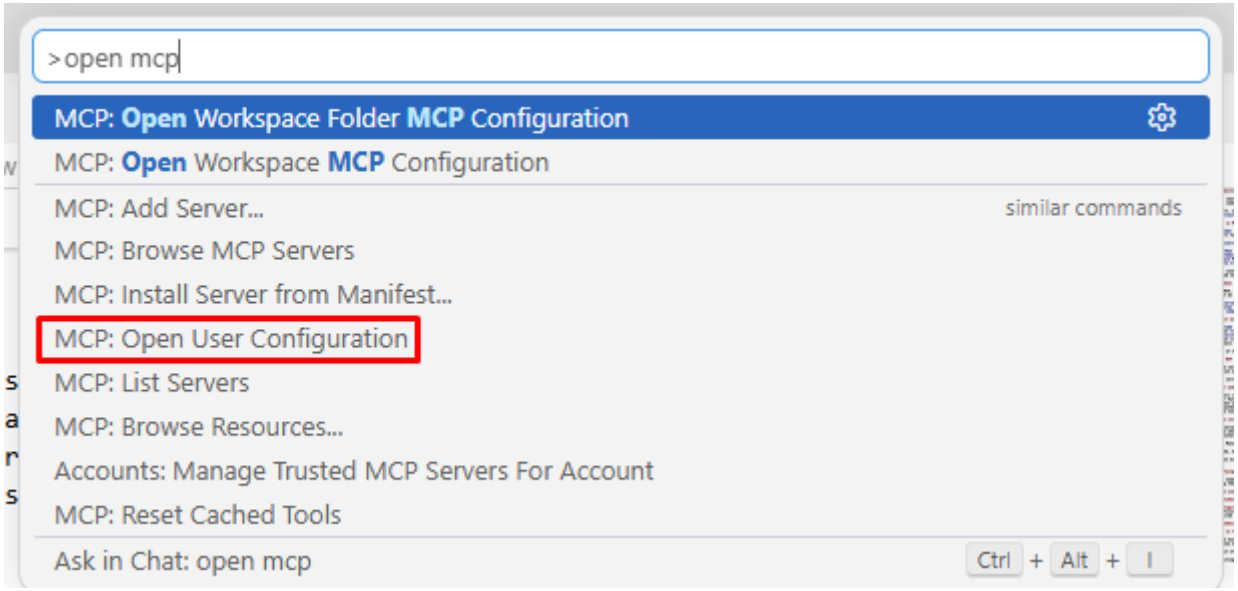
3.4.11 How to install MCP servers

The MCP servers are configured in a file called `mcp.json`. Depending on the server the actual configuration will differ - local servers need a path to where they are installed, some servers require authentication parameters etc. The documentation for the MCP server will tell you how to install it. Also, often there is an "Install in VSCode" button that allows you to add the configuration automatically.

Example: Manually installing the AFRY Foundations MCP server

In VS Code, bring up the command palette by pressing `CTRL+SHIFT+P`.

Type "open mcp" in the search bar, then select `Open User Configuration`



If the file is empty (containing only {}), replace the contents with:

```
{ "servers": { "foundation": { "url": "https://services.afry.cloud/mcp/foundation" }, } }
```

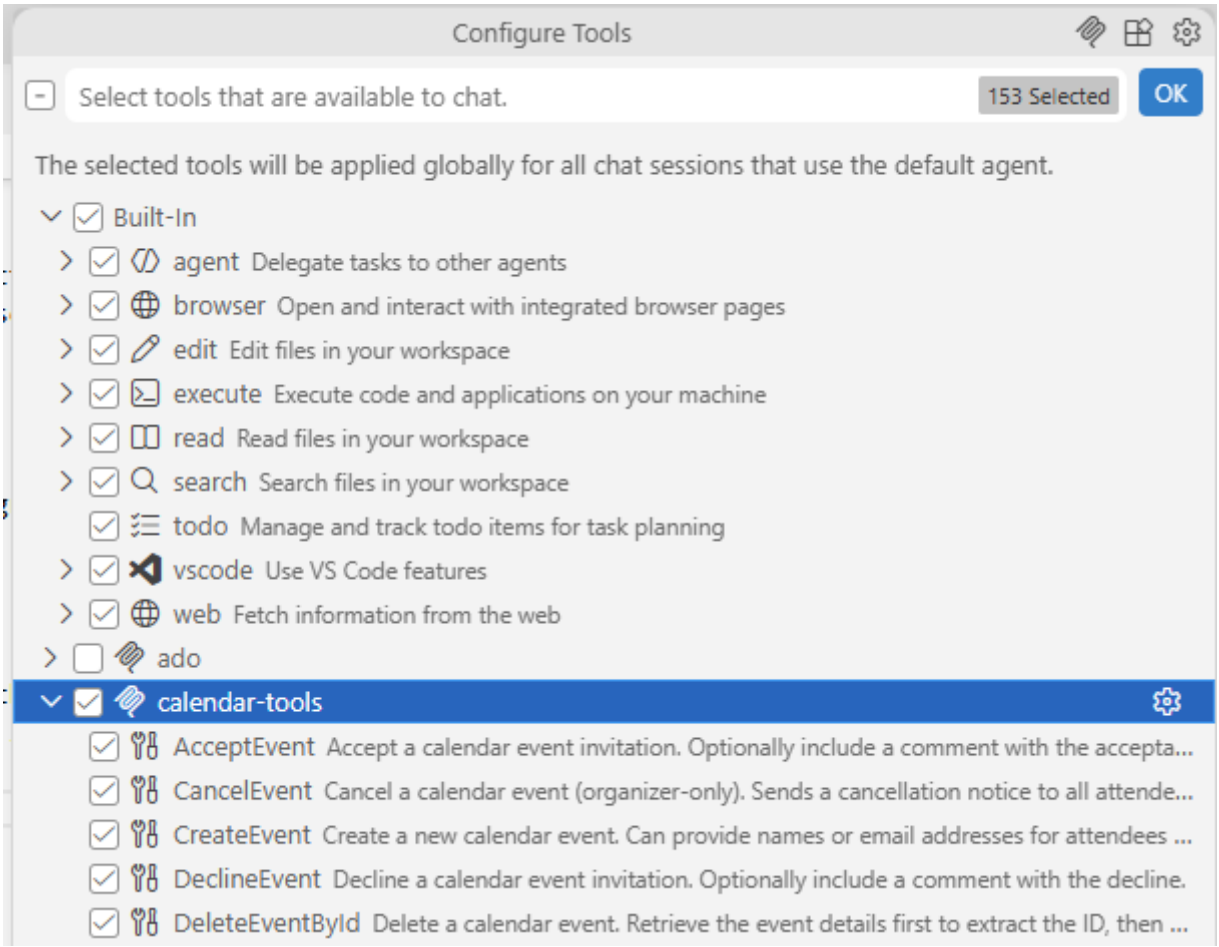
Verifying the installation

Open the Command Palette and run **MCP: List Servers**. Your server should appear with a green status indicator. If there's a problem, selecting the server shows the error log from the server process.

3.4.12 How to use MCP servers

The `Configure tools` button let's you select which tools are visible to the agent.

When the number of available tools grows large it becomes more difficult for the agent to know which tool to use in different situations. It is therefore good practise to select only the tools that are required for a specific task.



3.4.13 Where to find MCP servers

There are hundreds of MCP tools created by both companies and enthusiasts. Be aware though that when you connect an MCP server you do not really know where the data is going. Only install servers that you trust.

For a list of MCP servers that are developed and maintained by AFRY, visit []

For a listing of MCP public servers, visit [https://mcpservers.org/]

3.5 Getting Started: Skills

3.5.1 What is a skill?

Large Language Models are probabilistic. Everytime you ask something, the assistant will first reason on how to solve the task. Because of the randomness, the plan will never be exactly the same and oftentimes you need to correct the assistant when it derails or do things you don't want.

A **skill** is a text file that packages a set of instructions for Copilot. You invoke it by mentioning it in a prompt - Copilot reads the instructions and applies them to whatever you are working on.

With skills you can prepare the planning step for the agent - it will already have a script for it to follow, resulting in a more deterministic behaviour.

3.5.2 Skills vs. custom instructions

Both skills and custom instructions are markdown files that shape how Copilot behaves. The difference is *when* they apply:

	Custom instructions	**Skills**
When active	Always - loaded automatically in every conversation	On demand - only when you mention them
Scope	Sets the baseline behaviour for a project or user	Applies expert knowledge to a specific task
Example	"Always use TypeScript. Never use `var`."	"Apply AFRY brand colours and typography to this presentation."

Use custom instructions for standing rules. Use skills for specialised tasks you reach for occasionally.

3.5.3 Example: afry-brand-style-guide

The `afry-brand-style-guide` skill is a single markdown file that contains AFRY's full brand specification - primary colours, typography, layout rules, and logo usage. To apply it, you just reference it in a prompt:

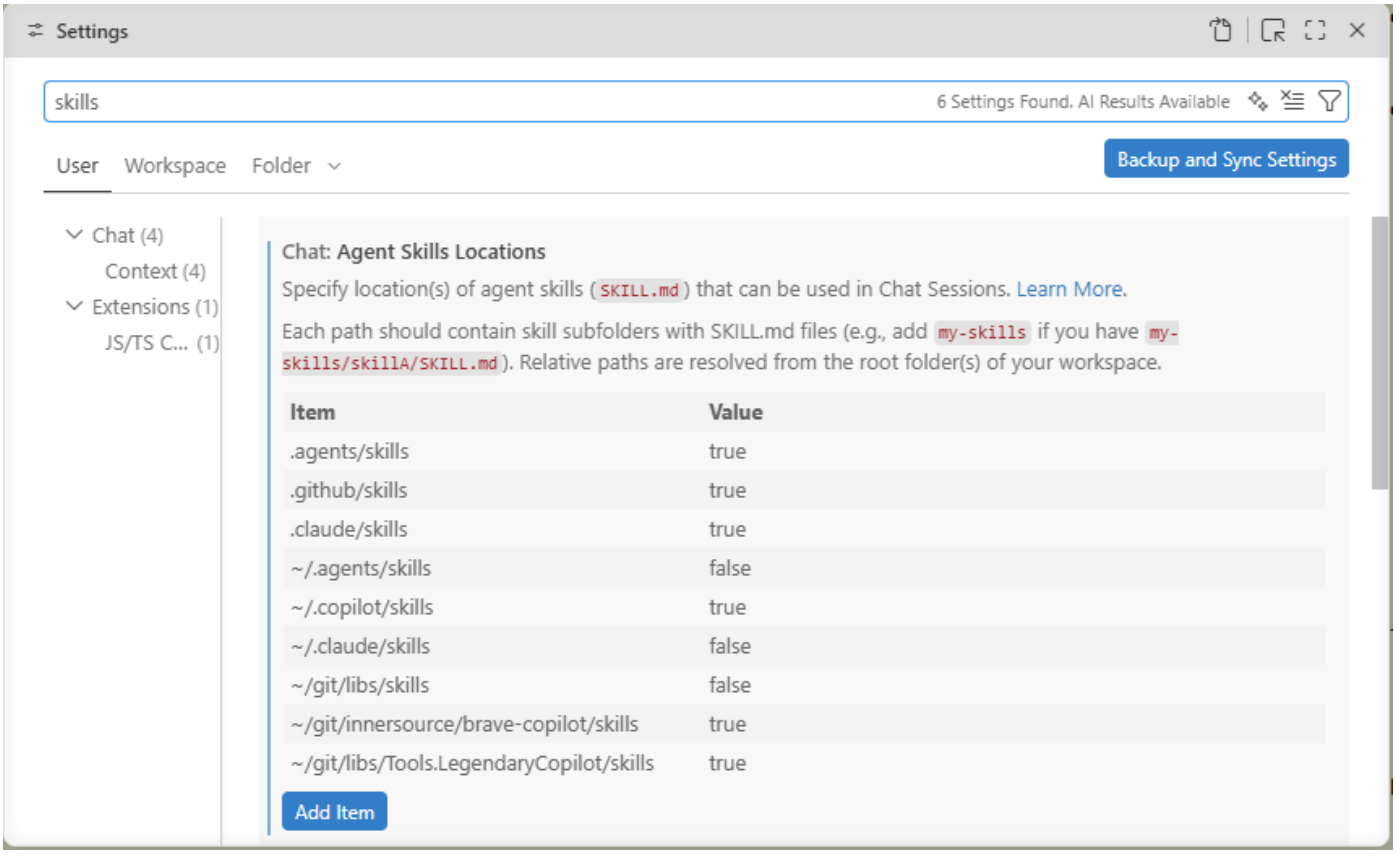
"Using the afry-brand-style-guide skill, create a styled HTML page for this content."

Copilot reads the skill, learns the colour palette and typography rules, and produces output that matches the AFRY brand - without you having to repeat any of those details yourself.

3.5.4 Where skills live and how Copilot finds them

Skills are plain `.md` files with a short section at the top (called the 'frontmatter'). VS Code finds them through its **customizations** system. You tell VS Code where to look by adding the folder to your settings.

1. Open the settings page from **File > Preferences > Settings**
2. Search for "skills" in the search bar



3. Add the folder where your skill folder is located.

Note The skills folder must somewhere inside your user folder, hence the ~ prefix in the path.

Once a skills folder is registered, every `.md` file in it becomes available as a skill. Copilot can also discover and suggest relevant skills automatically based on your prompt.

A skill file looks like this:

```
--- name: my-skill-name description: One-line summary of what this skill does and what phrases might trigger it. --- # My Skill #
```

The `description` field is important - Copilot uses it to decide whether the skill is relevant to a given prompt.

3.5.5 Brave Copilot - AFRY's shared skill library

Brave Copilot is AFRY's inner-source repository of ready-to-use skills, custom instructions, and agents. It contains skills for everyday AFRY work, such as

Skill	What it does
`afry-brand-style-guide`	Applies AFRY's official brand colours and typography
`powerpoint-sketch-writer`	Drafts an AFRY presentation as numbered slide markdown files
`powerpoint-deck-builder`	Builds a real `.pptx` from those slide files using the AFRY template

Full detail on Brave Copilot is covered in section 12.

3.5.6 Writing your own skill

You don't need to write any code to create a skill - just a markdown file. The fastest way is to let Copilot help you write it:

1. First work together with Copilot to achieve what you want the skill to do. You might write

Please extract all the project references from my time report in this Excel file, then write a comment for each project that summarizes what I've done.

- Copilot might generate Powershell scripts to extract the information
- The script doesn't work initially so you ask it to fix it.
- Copilot generates the summaries, but they are not in the language you prefer, etc. etc.
- When you finally arrive at something you are happy with, type

/create-skill Please create a skill that allows me to generate a summary of what I've done from a time report Excel file 3. Copilot will generate the skill and place it in your home folder (in ~.copilot/skills). 4. In a new chat, type Please summarize what I've done for each project 5. Copilot will find the skill and perform the steps 6. If there are mistakes in the instructions, you might need to go back to the earlier chat and tell Copilot what went wrong when you tried to use the skill.

Tips for good skills

- **Be specific in the description** - this is what Copilot uses to decide whether to invoke the skill automatically
- **One purpose per skill** - a skill that does one thing well is more reusable than one that tries to do everything
- **Use examples in the instructions** - showing Copilot an example of the desired output format is more reliable than describing it in abstract terms
- **Test by invoking directly** - reference the skill by name in a prompt and check the output before sharing it with your team

3.6 Getting Started: Agents

3.6.1 What is a custom agent?

When you open Copilot in VS Code, the default mode is the built-in **Agent** - a general-purpose AI that can use all tools and handle any task. That works well for most things.

A **custom agent** is a specialised version of that. It has a name, a persona defined in plain text, a specific set of allowed tools, and optionally a preferred model. You switch to it from the agents dropdown in the Chat panel, and from that point Copilot behaves according to its instructions.

Think of it as giving Copilot a specific job title for the session: instead of a generalist, it becomes a security reviewer, a deep-research analyst, or an AFRY PowerPoint builder. In a kitchen analogy, the agent is the Chef.

3.6.2 Custom agents vs. Agent mode

	Built-in Agent mode	**Custom agent**
Persona	Generic	Named specialist
Tool access	All tools	Restricted list you define
Instructions	None	The body of the `.agent.md` file
Model	Picker default	Optional override
Workflows	No	Handoffs between agents

3.6.3 The three built-in agents

VS Code ships three agents out of the box, and the differences between them illustrate exactly how tool and prompt choices shape behaviour.

Agent	**Purpose**	**Key tools available**
Agent	General-purpose agentic mode. Can edit files, run terminals, call MCP tools, and take multi-step actions.	All tools (edit, terminal, codebase, MCP, etc.)
Ask	Answer questions about your codebase without changing anything. Read-only and safe to run at any time.	Read and search only - no edit or terminal tools
Plan	Think through a complex task and produce a step-by-step plan before any code is written.	Read and search only - outputs a plan, does not execute it

The prompting is different too. Agent is instructed to act and iterate until the task is done. Ask is instructed to explain and answer. Plan is instructed to decompose and reason before committing to any action. The same underlying model, three very different behaviours - just from tool restrictions and system instructions.

Custom agents follow the same principle: you define the tools and the instructions, and you get a repeatable, predictable persona.

Use plain Agent mode for general coding and exploratory tasks. Use a custom agent when you want Copilot to consistently behave in a specific way - for example, only using read-only tools, always applying a certain review checklist, or always working within a defined set of MCP servers.

3.6.4 The `.agent.md` file

A custom agent is defined in a single markdown file with the `.agent.md` extension. The YAML frontmatter defines its configuration; the file body is the instruction text Copilot prepends to every prompt when that agent is active.

```
--- name: Security Reviewer description: Reviews code for security vulnerabilities. Read-only - no edits. tools: [read, search, c
```

Key frontmatter fields

Field	Description
`name`	Display name shown in the dropdown (defaults to filename)
`description`	Shown as placeholder text in the chat input
`tools`	Which tools this agent may use - omit to allow all
`model`	Preferred model (e.g. `GPT-4o (copilot)`) - optional
`agents`	Which other agents this agent may invoke as subagents
`user-invocable`	Set to `false` to hide from dropdown (subagent-only agents)

The instruction body can be up to 30,000 characters.

Where to put agent files

Scope	Location
Workspace	`.github/agents/*.agent.md`
Personal, all workspaces	`~/.copilot/agents/`

3.6.5 Invoking an agent

Direct invocation - you pick the agent yourself:

1. Open the Chat panel
2. Click the agents dropdown (shows all available agents)
3. Select the agent you want
4. Chat as normal - the agent's instructions are now active

Automatic invocation - Copilot picks the agent:

If the `runSubagent` tool is active, Copilot can automatically select and invoke a custom agent as a subagent when it decides the task is a good fit. You can disable this for a specific agent by setting `user-invocable: false` and `disable-model-invocation: true` in its frontmatter.

3.6.6 Subagents

A **subagent** is an agent invoked from inside another agent - used to delegate an isolated piece of work without interrupting the main conversation.

When the main agent encounters a subtask that benefits from a clean context (a web search session, a specialised review pass, a document generation step), it calls `runSubagent`, passing only the relevant prompt. The subagent runs independently and returns its result. This appears in chat as a collapsible tool call.

Example - a TDD agent that delegates to three specialists:

```
--- name: TDD tools: [agent, read, edit] agents: [Red, Green, Refactor] --- Implement features using Test-Driven Development. Use
```

Here `Red`, `Green`, and `Refactor` are themselves `.agent.md` files - each knows only its part of the workflow.

Brave Copilot's research agent is a real example of this pattern. The top-level `research` agent orchestrates the whole process - gathering context, deciding on perspectives, and then spawning `research.sub` subagents in parallel (one per research perspective) to do the actual web searching and report writing. A final `research.sub.review` subagent analyses the assembled report for gaps. Each subagent runs with its own focused instructions and a restricted tool set.

Subagent rules

- Nested subagents (subagents spawning further subagents) are off by default - enable with the `chat.subagents.allowInvocationsFromSubagents` setting
- Maximum nesting depth is 5
- A subagent's model cannot exceed the cost tier of the parent model

3.6.7 When to use a custom agent vs. plain Agent mode

Situation	Use
General coding, one-off tasks	Plain Agent mode
You want Copilot to always stay read-only for a task	Custom agent with restricted `tools`
You have a multi-step workflow you run repeatedly	Custom agent (or skill)
You want different Copilot personas for different domains	Custom agents
You want to delegate isolated subtasks cleanly	Subagents
You want reusable instructions without tool restrictions	Skill or custom instruction

3.6.8 Example: Brave Copilot's research agent

Brave Copilot ships a `research` agent that shows what custom agents and subagents can do together.

What it does: You give it a topic. It searches the web from multiple angles, writes a structured research report, and then reviews its own output for gaps - all without you having to prompt each step.

How it works:

1. `research` (main agent) - orchestrates the session. Takes your topic, proposes research perspectives, confirms with you, then spawns subagents.
2. `research.sub` (subagent) - web search worker. Runs once per perspective. Uses the **Foundation websearch** MCP tool to search the web, extracts facts, and writes its findings into the report file.
3. `research.sub.review` (subagent) - quality reviewer. Reads the assembled report and flags unsubstantiated claims, missing information, and gaps.

The `research.sub` and `research.ask` agents have `user-invocable: false` - they never appear in the dropdown. They only exist to be called by the orchestrating agent.

Full detail on Brave Copilot - including how to install it and what agents and skills it contains - is in section 12.

4. AFRY Tools

4.1 Brave Copilot

4.1.1 What is Brave Copilot?

Brave Copilot is AFRY's inner-source repository of skills, custom agents, and instructions for GitHub Copilot (and other AI agents). It packages AFRY-specific domain knowledge - brand standards, Azure patterns, DevOps workflows, presentation templates, vibe coding pipelines - into reusable files that any AFRY employee can plug into their VS Code setup.

Instead of every team rediscovering how to do the same things, Brave Copilot is the shared library that grows as colleagues contribute.

The repository lives at: `github.com/afry-innersource/brave-copilot`

4.1.2 Two types of content

Brave Copilot contains two kinds of customisation files:

Type	File format	How to invoke	What it does
Skills	`SKILL.md` in a named folder	Mention the skill name in a prompt	Gives Copilot a detailed script for a specific task
Custom agents	`.agent.md`	Pick from the agents dropdown	Defines a named Copilot persona with restricted tools and instructions

Skills and agents (and MCP tools) work together - an agent can invoke skills and tools as part of its workflow.

4.1.3 Available skills

Skill	What it does
`afry-brand-style-guide`	Applies AFRY's official colours, typography, and layout rules to any artefact - presentations, web UIs, documents
`powerpoint-sketch-writer`	Drafts a presentation as numbered markdown slide files - a planning step before building the real deck
`powerpoint-deck-builder`	Builds a real AFRY-branded `.pptx` from a markdown sketch using the PowerPoint MCP server - requires the **PowerPoint MCP server** (see section 13)
`vibe-create-web-app`	Scaffolds a React + Vite + Tailwind CSS web app - requires**Node.js** (see section 5)
`vibe-add-pwa`	Makes an existing Vite app installable as a PWA (phone, tablet, desktop)
`vibe-add-backend`	Adds an Azure Functions API backend to a Vite app - requires**Node.js** and **Azure CLI**
`vibe-add-llm`	Provisions Azure OpenAI and wires up a `/api/chat` streaming endpoint - requires **Azure CLI**
`vibe-add-auth-for-swa`	Adds AFRY-only Entra ID sign-in to a deployed Azure Static Web App - requires**Azure CLI** and an App Registration
`vibe-deploy-app`	Detects your project type and deploys it to Azure (orchestrator) - requires**Azure CLI** and **GitHub CLI**
`vibe-add-remote-store`	Provisions Cosmos DB and wires it into the Functions backend - requires**Azure CLI**
`vibe-add-entity`	Generates a TypeScript type, repository, and API endpoint for a new data entity
`vibe-add-data-access`	Changes who can read or write an entity: user-only, public, or group-shared
`vibe-add-local-store`	Adds browser-side storage (localStorage or IndexedDB) - no Azure required

The vibe coding skills are designed to work as a chain - create → add backend → add auth → add data → deploy. Each step extends the previous one. Before running any vibe coding skill, make sure the software listed in section 5 is installed.

4.1.4 Available agents

`research` - Deep-research agent. You give it a topic; it proposes research perspectives, confirms with you, then runs parallel web searches (using the **Foundation websearch MCP tool**) and assembles a structured report. A final review pass checks the report for gaps and unsupported claims.

To use it, select `research` from the agents dropdown and describe what you want to research. The agent handles the rest - you can follow along as it works.

4.1.5 How to install

Prerequisites

You need **Git** installed before you can clone the repository. Git installation is covered in section 5. Individual skills may also require additional software such as Node.js, Azure CLI, or specific MCP servers - check the notes in the skills table above and section 13 for MCP setup.

Skills and agents must be placed inside your home directory (`C:\Users\YourName\` on Windows, `~/` on Mac/Linux). VS Code does not pick them up from other locations.

Step 1 - Clone the repository into your home directory

1. Open VS Code
2. Press `Ctrl+Shift+P` to open the Command Palette
3. Type `Git: Clone` and select it
4. Enter the repository URL:

   `https://github.com/afry-innersource/brave-copilot.git`

5. When asked where to clone, choose a folder inside your home directory - for example `C:\Users\YourName\repos\` - and confirm

VS Code will clone the repository and offer to open it. You can close it afterwards; you do not need to keep it open to use the skills.

Step 2 - Register the skills folder in VS Code

As described in section 10, open VS Code Settings (`Ctrl+,`), search for **skills**, and add the path to the cloned `skills/` folder:

- Under **Chat: Agent Skills Locations**, add the path to `brave-copilot/skills`
- Under **Chat: Prompt Files Locations**, add the same path

For example: `C:\Users\YourName\repos\brave-copilot\skills`

Step 3 - Copy the agents to your home folder

Copy the `.agent.md` files from the cloned `agents/` folder into:

```
C:\Users\YourName\.copilot\agents\
```

Create the `.copilot\agents\` folder if it does not exist yet. VS Code looks in this location automatically for user-scoped agents.

Verify

Type `/` in the Copilot chat input - the Brave Copilot skills should appear as slash commands. Open the agents dropdown and check that `research` is listed.

4.1.6 Using a skill

Once installed, invoke a skill by mentioning its name in a prompt:

"Using the `afry-brand-style-guide` skill, apply AFRY styling to this HTML page."

"Run the `vibe-create-web-app` skill to scaffold a new app for tracking team decisions."

You can also type `/` followed by the skill name to use it as a slash command.

4.1.7 Contributing

Brave Copilot is an inner-source repository - any AFRY employee can contribute a skill.

1. Fork the repository
2. Create a folder under `skills/` with your skill name (e.g. `skills/my-skill/`)
3. Write a `SKILL.md` with YAML frontmatter (`name`, `description`) and your instruction body
4. Open a pull request with a short description of what the skill enables

The skill does not need to be perfect on the first attempt. A useful rough skill is more valuable than no skill at all. Copilot can even help you write the `SKILL.md` file - after a successful interaction, type `/create-skill` in the chat input and Copilot will draft the skill file from the conversation.

4.1.8 Community

Brave Copilot is a living library. AFRY teams across disciplines are building their own skill collections:

- **Data and AI Platform** - MCP server automation, pipeline orchestration, Azure search and index configuration
- **Finance** - Sprint reports, Azure DevOps work item creation, team status summaries
- **Vibe Coding** - The full create → deploy → auth → data → LLM chain

If your team has a workflow that you have taught Copilot once, it probably belongs in Brave Copilot so the rest of AFRY can benefit too.

4.2 MCP Servers

AFRY maintains several Model Context Protocol (MCP) servers that extend AI capabilities for specialized domains. These servers provide tools and resources accessible through the MCP protocol for integration with AI assistants and other applications.

4.2.1 Foundation MCP

The Foundation MCP provides core utilities for user profile management, organizational search, and general platform integration. This server acts as a foundation layer for accessing AFRY's organizational data and services.

Key capabilities:

- User profile retrieval and management
- Organizational search and lookup
- General platform utilities

Repository: Service.MCP.Foundation

4.2.2 Local File Transfer MCP

The Local File Transfer MCP allows Copilot to send and receive files from other MCP tools, such as downloading user profile images from the Foundation tools. This server can be installed from the Application Kiosk.

Key capabilities:

- Local file operations (read, write, delete)
- File transfer between systems
- Secure file access management

4.2.3 PowerPoint MCP

The PowerPoint MCP provides comprehensive automation capabilities for Microsoft PowerPoint. This server enables programmatic creation, modification, and management of PowerPoint presentations, including support for AFRY template layouts and styling.

Key capabilities:

- PowerPoint file creation and editing
- Slide management and layout application
- AFRY template integration
- Formatting and styling automation

Repository: mcp-powerpoint

4.2.4 DocPipeline MCP

The DocPipeline MCP handles document processing workflows and pipeline automation. This server coordinates multi-stage document processing operations, enabling complex document transformation and analysis workflows.

Key capabilities:

- Document pipeline orchestration
- Multi-stage processing workflows
- Document transformation automation

4.2.5 DocToAI MCP

The DocToAI MCP converts documents into AI-ready formats and coordinates document-to-AI workflows. This server bridges document repositories with AI services, enabling intelligent document analysis and processing.

Key capabilities:

- Document format conversion
- Document-to-AI pipeline integration
- Intelligent document processing
- AI analysis coordination

5. Vibe Coding

5.1 Vibe Coding: GitHub and Git

Before you build and deploy your first app, you need two things in place: somewhere to store your project, and a way to track your changes. That is what Git and GitHub give you.

5.1.1 Have you ever done this?

Look at this folder:

```
report.docx report_final.docx report_final_v2.docx report_final_FINAL.docx report_final_FINAL_reviewed.docx
```

If that looks familiar - you have already been doing version control. Just manually, and without much safety net.

Git solves this. Instead of saving copies with new names, you:

- Keep one file (or one project folder)
- Let Git automatically track every change you make
- Give each saved state a short description ("Added login page", "Fixed the export bug")
- Compare any two versions, or roll back to any point in history - instantly

5.1.2 Then there is collaboration

Now imagine you and a colleague are both editing the same PowerPoint presentation in M365. You can both make changes at the same time, and OneDrive syncs everything automatically - no committing, no pushing, no thinking about it. It just works. That is great for a single file.

But notice: it only helps with that one file. If your project is a folder full of related files - a presentation, a data file, a script that generates the charts - M365 does not know they belong together. You are back to the `report_final_FINAL.docx` problem, just spread across multiple files instead of one.

Git is designed for exactly this. It treats an entire project folder as a single unit, tracks changes across every file together, and lets you commit them as a group with a single description of what changed and why. You also decide when to save a snapshot - which means your history reflects meaningful moments ("Updated the chart data and

regenerated the slides") rather than every individual keystroke. The trade-off is that it requires a little more deliberate action: you commit, you push. But you get full control over your project history, across every file, with none of the naming chaos.

5.1.3 Git vs GitHub

People use the names interchangeably, but they are different things:

	What it is
Git	The tool installed on your computer that tracks changes and manages history
GitHub	The cloud platform that stores repositories and adds collaboration, CI/CD, and issue tracking

Git works entirely on your own machine. GitHub is where you push your work so others can access it - and where the vibe coding deploy pipeline picks it up.

5.1.4 Key concepts

Term	What it means
Repository (repo)	Your project folder, with its full change history baked in
Commit	A named save point - "Added login page" rather than "save3"
Branch	Your own working copy of the project - experiment freely without touching the main version
Push	Send your local commits up to GitHub
Pull	Download the latest commits from GitHub to your machine
Pull request	A request to merge your branch into the main version - others can review before it goes in

You do not need to memorise all of these before you start. The vibe coding trail will walk you through each one in context.

5.1.5 AFRY's GitHub setup

AFRY uses **GitHub Enterprise Managed Users (EMU)**. This means your GitHub account is created and managed by AFRY through Entra ID - it is not a personal account you own. You log in with your AFRY credentials, and the account is scoped entirely to the AFRY enterprise.

Personal repositories

You can create repositories under your own user account, but they are always **private**. This is an EMU restriction - no public content is allowed. Personal repos are fine for experimenting, but they come with a key limitation for vibe coding: **you cannot set up GitHub Actions for automated deployment from a personal repo**. For anything you want to publish or deploy with CI/CD, you need an organisation repo.

Organisations

AFRY has a set of GitHub organisations - one for each division, plus a shared one for cross-company tools:

Organisation	What it is for
AFRY-Inner-Source	Tools and resources shared across all of AFRY
AFRY-Energy	Energy division
AFRY-Global-IT	Global IT
AFRY-IDS	IDS division
AFRY-Infrastructure	Infrastructure division
AFRY-ManagementConsulting	Management Consulting
AFRY-ProcessIndustries	Process Industries
AFRY-X	AFRY X

You will only see the organisations you have been added to. Access is managed by each organisation's owners.

Visibility inside an organisation

When you create a repository in an organisation, you choose between two visibility levels:

Visibility	Who can see it
Internal	Every AFRY employee with a GitHub account - across all organisations
Private	Only you and the collaborators you invite

Use **Internal** for anything that should be open to all of AFRY (like Inner-Source content). Use **Private** when you want to control exactly who has access - you can still invite specific collaborators to a private repo. Either way, no one outside AFRY can see it.

For your own vibe coding projects, you will typically create a **private** repository in the organisation that matches your division. This gives you GitHub Actions support for deployment, while keeping the code visible only to your team.

5.1.6 The six Git commands you actually need

You do not need to memorise more than these to get through the vibe coding trail:

Command	What it does	When you use it
`git clone <url>`	Download a repository from GitHub to your machine	The first time you work on a project
`git status`	Show what has changed since your last commit	Constantly - before you commit
`git add .`	Stage all changed files for the next commit	Before every commit
`git commit -m "message"`	Save a named snapshot of your staged changes	Whenever you reach a logical stopping point
`git push`	Send your commits to GitHub	After committing - to back up and share
`git pull`	Get the latest changes from GitHub	When starting a session or collaborating

5.1.7 Use Copilot as your Git tutor

Git has a reputation for being confusing - and honestly, that reputation is not entirely unfair. Branches, merge conflicts, rebasing, detached HEAD states... there is a lot to get wrong, and the error messages are not always helpful.

Here is the good news: GitHub Copilot is excellent at Git. It understands the concepts deeply, can explain them in plain language, and can walk you through any situation you get stuck in. That makes it an ideal tutor - not just a command generator, but something you can actually have a conversation with.

Ask it to explain things:

"Explain how Git branches work, as if I have never used version control before."

"What does 'HEAD' mean? ELI5 please."

"What is "merge"? What is the difference between "merge" and "rebase", in simple terms?"

Ask it to help you do things:

"I want to create a new branch, make some changes, and then merge it back into main. Walk me through the steps."

"I ran git pull and now I have a merge conflict. What do I do?"

"I committed to the wrong branch. How do I move that commit to the right one?"

Ask it to review what happened:

"Show me how to see the history of this file and who changed what."

"I think I messed something up. How do I see what changed since my last commit?"

Let it run the commands for you - but ask it to explain:

One of the best things AI has brought to programming is automatic commit message generation. Instead of writing "fixed stuff" for the hundredth time, Copilot looks at exactly what changed and writes a meaningful, descriptive commit message for you. This alone is worth using it for.

You can go further and ask Copilot to handle the whole workflow - staging, committing, pushing - and have it explain each step as it goes. This is one of the fastest ways to build real understanding:

"Stage all my changes, write a commit message based on what I changed, commit, and push. Explain each command before you run it."

You get the work done and learn Git at the same time. After a few sessions of watching Copilot narrate its way through a push, the commands start to feel natural.

The more you ask, the more you learn - and the faster you build up intuition for how Git actually thinks. Treat it as a conversation, not a search engine.

VS Code also has a built-in **Source Control panel** (`Ctrl+Shift+G`) if you prefer a visual interface for staging, committing, and syncing - and the **Git Graph** extension gives you a visual diagram of your branch history. But even if you use those tools day to day, Copilot is the fastest way to understand what is happening under the hood.

5.1.8 Git vs GitHub CLI - two different tools

This trips people up. There are two command-line tools with similar names:

Tool	What it is	When you need it
`git`	Version control - tracks changes, creates commits, syncs with GitHub	Always - needed for every step of vibe coding
`gh` (GitHub CLI)	Automates GitHub platform actions from the terminal - creating repos, setting secrets, triggering deployments	Only when deploying websites or setting up CI/CD (sections 16.4+)

Both are covered in the installation instructions. You need `git` from the start. You can install `gh` later when you reach the deploy step - the deploy skill will prompt you to run `gh auth login` the first time.

5.1.9 Creating your first repository

1. Go to github.com and sign in with your AFRY account
2. Click the `+` button in the top right and select **New repository**
3. Give it a name, set visibility (Private or Internal), and click **Create repository**
4. Copy the repository URL
5. In VS Code, press `Ctrl+Shift+P`, type `Git: Clone`, and paste the URL

5.1.10 Learning more

This chapter gives you enough to follow the vibe coding trail. If you want to go deeper:

- GitHub Git Handbook - a concise official introduction
- VS Code Source Control docs - everything the built-in Git UI can do

You can also just ask Copilot. Any Git concept or command is fair game:

"Explain what a merge conflict is and how to resolve one."

"What is the difference between git fetch and git pull?"

Copilot will explain it in plain English, or walk you through fixing a specific problem in your repository.

5.2 Vibe Coding: Introduction

5.3 What is vibe coding?

Vibe coding is a way of building software by describing what you want, and letting an AI assistant do the actual coding.

You say: "I want a web page where I can type in a decision we made as a team, save it, and search through it later."

Copilot builds it.

You do not need to know how to write code. You need to be able to describe what you want clearly - the same skill you use when writing a requirements document or briefing a colleague. The AI figures out the technical details. You stay in charge of what gets built and why.

This is a real shift. Until recently, building a web app and putting it on the internet was something only developers could do. Now anyone willing to work with an AI assistant can do it.

5.3.1 The gap between a prototype and a real app

Here is the thing though: building something that runs on your own computer is actually the easy part.

The hard part has always been everything that comes after:

- **Deployment** - how do you make the app available so your colleagues can use it, not just you?
- **Authentication** - how do you make sure only AFRY employees can log in?
- **Data** - how do you store things so they are not lost when you close the browser, and so the whole team can see each other's entries?
- **AI features** - how do you connect your app to a language model?

Each of these requires knowledge that takes years to build up. You need to understand cloud platforms, deployment pipelines, identity systems, databases, and API design. A beginner trying to navigate all of this alone will quickly get stuck.

This is exactly the problem the **Brave Copilot vibe skills** were designed to solve.

5.3.2 The vibe coding skills

The vibe coding skills are a set of ready-made instructions that Copilot follows to handle the hard parts for you. Each skill knows how to do one specific thing - deploy an app, add login, connect a database - and handles all the technical decisions that would otherwise block a beginner.

You describe what you want. The skill knows how to build it. Copilot does the work.

Skill	What it does
vibe-create-web-app	Creates a new web app - a blank canvas with a modern, professional foundation
vibe-deploy-app	Publishes your app to the internet so anyone can reach it at a real web address
vibe-add-auth-for-swa	Adds login, so only AFRY employees can access your app
vibe-add-local-store	Saves data in the browser - no server needed, but data stays on one device
vibe-add-remote-store	Adds a shared database - data is stored in the cloud and visible to every user
vibe-add-llm	Connects your app to an AI language model so it can understand and generate text

Together these skills take you from "I have an idea" to "I have a live app with login and a database" - step by step, without needing to become a software engineer first.

5.3.3 What is Azure?

When you deploy an app, it needs to live somewhere. That somewhere is **the cloud** - a network of powerful servers, managed by a technology company, that you can rent space on.

AFRY uses **Microsoft Azure** as its cloud platform. Azure is where AFRY's apps, services, and data run. When you publish your vibe coding project, it goes to Azure too - inside a **resource group** that belongs to your team or division.

A resource group is simply a container in Azure - a named area where your app and its supporting services live together.

Think of it like this: your laptop is your workshop. Azure is the building where you rent a unit. Your resource group is your unit - your own space, inside AFRY's building, where you put your app.

You do not need to learn how Azure works in depth. The vibe coding skills handle the Azure setup for you. But it helps to know the name, because you will see it come up in the steps.

5.3.4 Costs

The trail creates real Azure resources, and real resources have real costs. The numbers vary depending on how far you go:

- **Steps 16.3 and 16.4 only** (create and deploy, no login): the Static Web App Free tier costs **€0**, plus minimal Cosmos DB and AI fees - typically **under €2 per month** in total.
- **Full trail including step 16.5** (add AFRY login): authentication requires upgrading to the Standard tier at **~€8 per month**, bringing the total to roughly **€9-10 per month**.

A full breakdown with per-service numbers, example calculations, and instructions for setting a budget alert is in the prerequisites chapter.

5.3.5 What is the Azure CLI?

When the vibe coding skills set up your Azure resources, they need a way to talk to Azure from your computer. That is what the **Azure CLI** does - it is a small tool you install once, and it gives Copilot a way to create and configure Azure services on your behalf, from the terminal.

The command is `az`. You will mostly see it in the background - Copilot runs it for you as part of the deployment skill. The main thing you need to do is log in once with `az login`, which opens a browser and connects the tool to your AFRY account.

Installation is covered in the prerequisites chapter.

5.3.6 The example app: Decision Log

Throughout this trail we build the same app, one step at a time. The app is called **Decision Log** - a simple tool for recording and searching the decisions your team makes.

It is an example that touches every capability in the trail:

Step	What we add	What changes
16.3 - Create	The app itself	A web page where you can type and save decisions; runs on your own computer
16.4 - Deploy	Publishing	The app goes live at a real URL; anyone with the link can reach it
16.5 - Auth	Login	Only AFRY employees can sign in; each decision is tagged with who wrote it
16.6 - Data	Shared database	Decisions are stored in the cloud; the whole team can see and search them
16.7 - LLM	AI	The app can surface past decisions relevant to a new question; it can also draft a decision record from a rough description

By the end of the trail you have a real, working, deployed app with login, shared data, and AI built in. You can keep it, extend it, and use it with your actual team.

5.3.7 Before you start

Work through the prerequisites chapter first. It has a checklist of everything you need installed and set up before the first step.

And remember: you do not need to understand every technical detail as you go. That is what Copilot is there for. Ask it to explain anything that is unclear - it is a good teacher as well as a good builder.

5.4 Vibe Coding: Prerequisites

Before you start the vibe coding trail, you need a few things in place.

5.4.1 The checklist

#	What you need	Why	Status
1	**GitHub Copilot**	The AI that does the building	✅ Required
2	**VS Code**	The editor where everything happens	✅ Required
3	**Git**	Version control - stores your project and powers the deployment pipeline	✅ Required
4	**Node.js**	The runtime that your web app and build tools need	✅ Required
5	**Azure CLI (`az`)**	Lets Copilot create and configure Azure resources on your behalf	✅ Required
6	**GitHub CLI (`gh`)**	Lets Copilot push to GitHub and set up the deployment pipeline	✅ Required
7	**An Azure resource group**	The container in Azure where your app will live	✅ Required
8	**An App Registration**	Needed only for step 16.5 (adding login)	⏳ Can wait
9	**Brave Copilot installed**	The skills library that contains the vibe coding skills	✅ Required

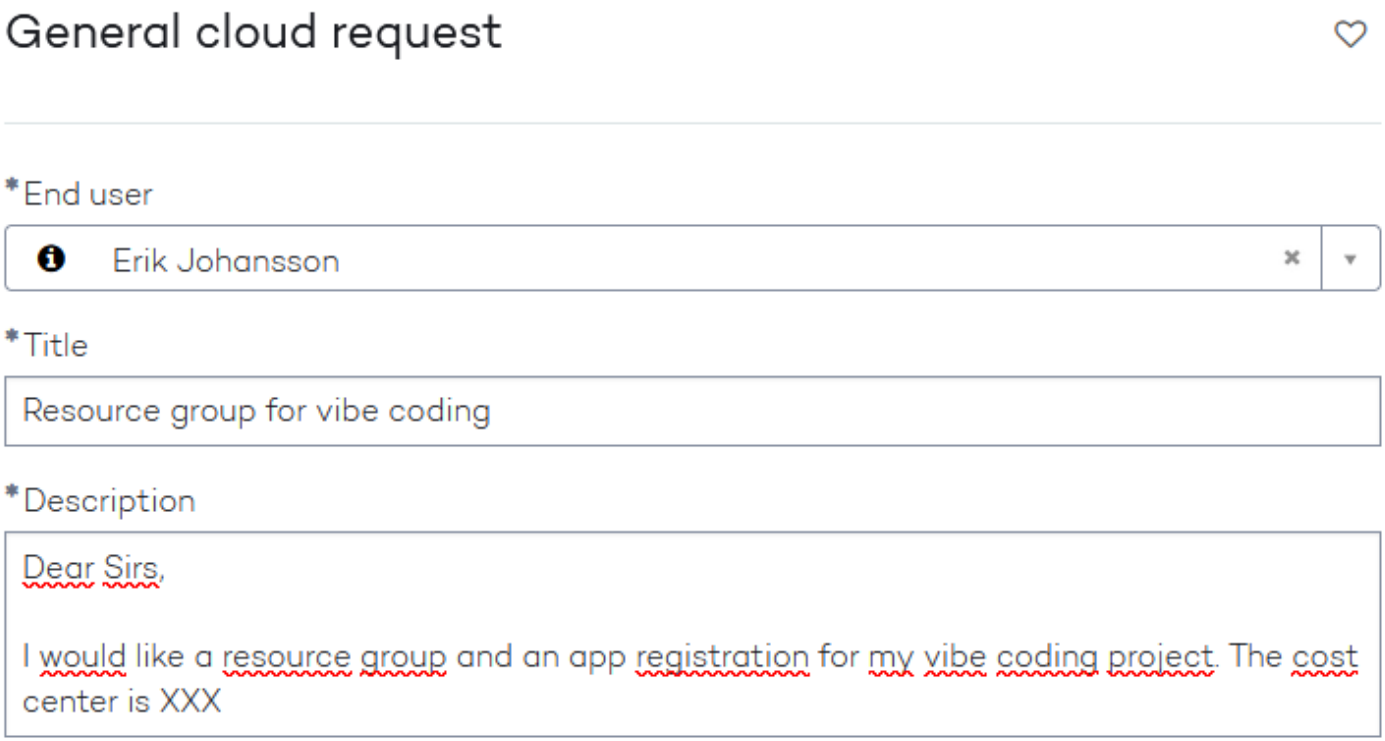
The sections below walk through each item.

5.4.2 1-6: Software

These are covered in the software you need chapter. If you have already worked through that page, you are good. If not, do it now - the installation of some tools (particularly GitHub Copilot) requires manager approval, which can take time. Other apps will need permission from AFRY IT as they are not yet available in the application kiosk.

5.4.3 7: Azure resource group

A resource group is the container in Azure where your app and its services will live. You need one before you can deploy anything. The resource group will be need to be tagged with your **name** and with a **cost center**. The cost center is required because some features - such as talking to a LLM or storing data in a shared database - will be billed monthly. Read more [here]



1. Raise a "General Cloud Request" through the AFRY IT service desk
2. Say: "I would like a resource group and an app registration for my vibe coding project. The cost center for the resource group is XXX"

You will receive the resource group name and the app registration name once they have been created.

5.4.4 9: Brave Copilot

The vibe coding skills live in the Brave Copilot repository - AFRY's inner-source library of Copilot skills.

If not installed, follow the installation instructions in the Brave Copilot chapter.

5.4.5 Costs

The vibe coding trail creates real Azure resources, and real Azure resources have real costs. The good news: for a small app used by a limited number of people, the monthly bill is very low - often just a few euros.

Here is what each step costs:

Static Web App (step 16.4 - deploy)

The **Free tier** covers most use cases and costs nothing:

Tier	Monthly cost	What you get
Free	€0	100 GB bandwidth, up to 2 custom domains
Standard	~€8	Same, plus custom auth, extra features and SLA

For a small internal tool with no login required, the free tier is sufficient. If you go through step 16.5 (adding AFRY login), you will need to upgrade to the **Standard tier** - custom authentication providers are not supported on the Free tier.

Shared database (step 16.6 - Cosmos DB serverless)

Cosmos DB serverless charges only for what you actually use - there is no minimum monthly fee and no charge when idle. The billing unit is the **Request Unit (RU)**, which is a measure of the computing power used by each database operation.

What you pay for	Price
Request Units consumed	~€0.23 per million RUs
Storage	~€0.23 per GB per month

Example: A small team app with 1 000 operations per day (reads, writes, searches):

- ~30 000 operations/month × 5 RUs each = 150 000 RUs
- Cost: less than **€0.05 per month**
- Storage for a few thousand records: roughly **€0.10 per month**
- **Total: under €0.20 per month**

Even at 10x that usage - a busier app, more records - you are looking at **€1-2 per month**.

AI features (step 16.7 - Azure OpenAI)

Azure OpenAI charges per **token** - roughly three-quarters of a word. Both what you send to the model (input) and what it sends back (output) count.

Model	Input	Output
GPT-4.1	~€1.84 per million tokens	~€7.36 per million tokens
GPT-4o	~€2.30 per million tokens	~€9.20 per million tokens

Example: 100 conversations per month, each about 500 tokens (a medium-length exchange):

- Total tokens: ~50 000
- Cost with GPT-4.1: **~€0.20**

Even 1 000 conversations per month - a heavily used tool - costs **~€2 per month**.

Total for the full trail

For a small internal team tool running all features (deployed, with auth, shared database, and AI):

Component	Estimated monthly cost
Static Web App (Free tier)	€0
Cosmos DB (low traffic)	< €0.50
Azure OpenAI (light usage)	< €1.00
Total	**< €2 per month**

Costs scale with usage. If the app becomes popular and usage grows significantly, they can increase - but still typically remain modest for internal tools.

Setting a budget and watching costs

Azure makes it straightforward to set a spending limit and get notified before you exceed it.

To set a monthly budget:

1. Open the Azure Portal and navigate to your resource group (search for the name of the resource group as given to you by AFRY IT)
2. In the left menu, go to **Cost Management** - **Budgets**
3. Click **Add** and set a monthly amount (e.g. €10)
4. Add alert thresholds - for example, send an email at 80% and 100%
5. Enter your email address as the recipient

Azure will email you when spending approaches your limit. Note that the budget alert is a **notification only** - it does not automatically stop the resources. To enforce a hard stop, you would need to involve AFRY IT.

To see what has been spent:

In your resource group, go to **Cost Management** - **Cost analysis**. You will see a breakdown by service, a daily chart, and a forecast for the rest of the month based on current usage. It is a clear, readable dashboard - no expertise needed.

Azure prices shown here are approximate EUR equivalents based on USD list prices. EUR amounts vary slightly month to month due to exchange rate updates. Use the Azure pricing calculator for precise current figures.

5.4.6 Final check

Once everything is in place, open a terminal in VS Code and run these four commands one by one. Each should return a version number - not an error:

```
git --version node --version az --version gh --version
```

When all four pass, you are ready to start 16.2a - Create your repository.

5.5 Vibe Coding: Create Your Repository

🎬 Video

Before you write a single line of code, you need a home for it. This chapter walks you through creating a GitHub repository for the Decision Log app and cloning it to your computer.

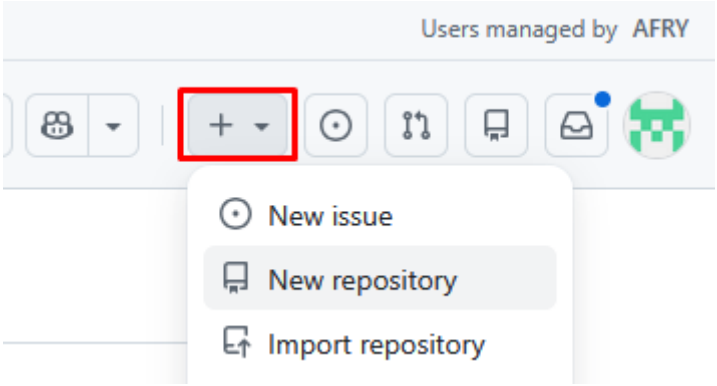
The whole thing takes about five minutes.

5.5.1 What you are setting up

You will create a repository called `yourname-decision-log` in an AFRY GitHub organisation, with **Private**visibility. Internal means every AFRY employee with a GitHub account can browse it - good for a learning project that others can learn from too. The name prefix is there because many people on the trail will build the same app, and name collisions on GitHub are not allowed.

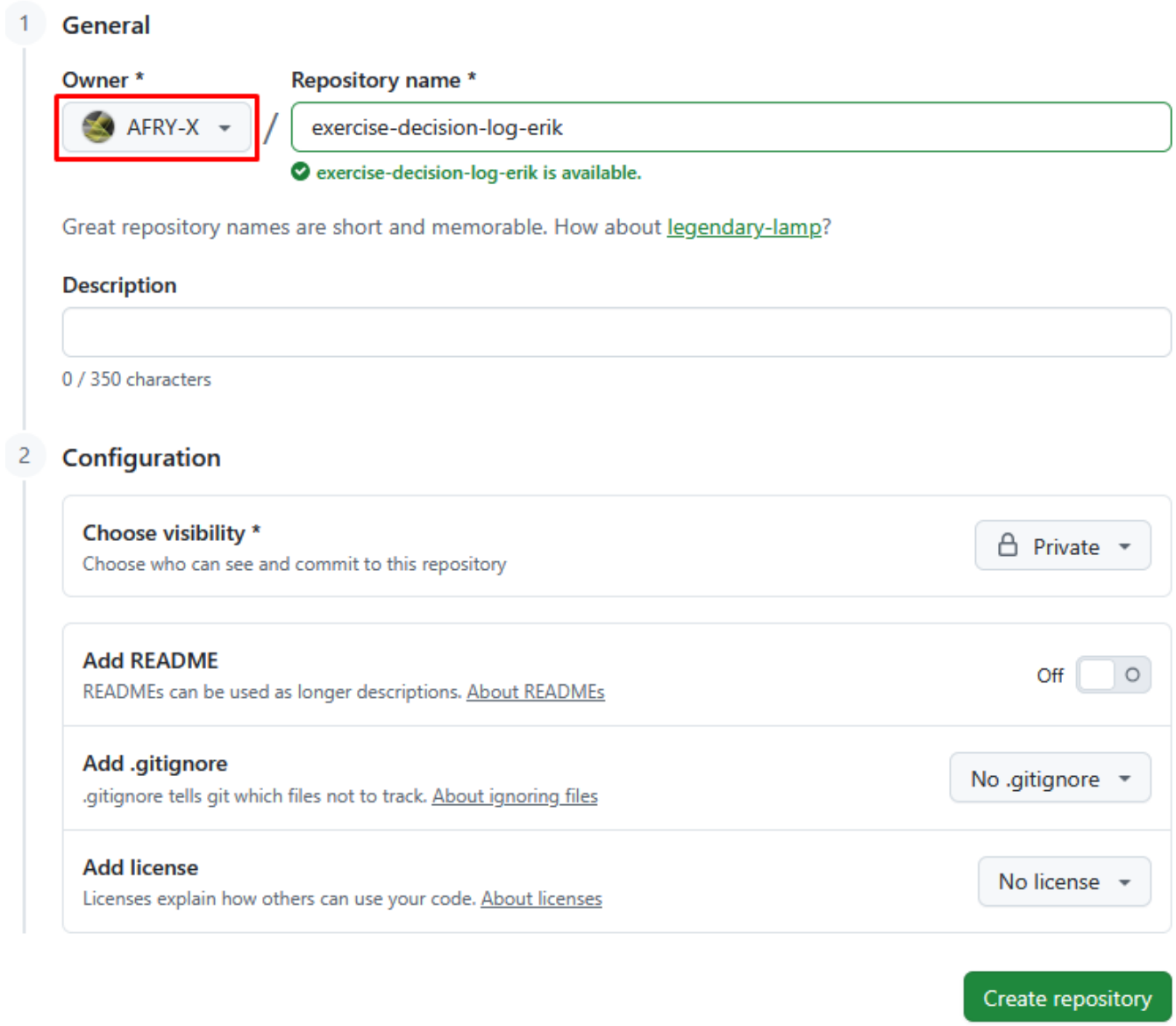
5.5.2 Step 1 - Create the repository on GitHub

1. Open github.com and sign in with your AFRY account



2. Click the **+** icon in the top-right corner and choose **New repository**
3. Set the **Owner** to your division's AFRY organisation (not your personal account - personal repos cannot use GitHub Actions for deployment). If you are unsure which org to use, check the table in 16.0 - GitHub and Git
4. Set the **Repository name** to `exercise-decision-log-yourname`, replacing `yourname` with your first name in lowercase - for example `exercise-decision-log-erik`
5. Set **Visibility** to **Private**

6. Leave everything else at its default

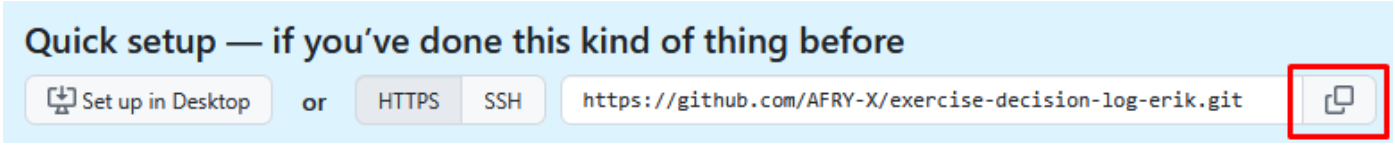


7. Click **Create repository**

5.5.3 Step 2 - Copy the repository URL

Once the repository is created, you will land on its main page.

1. Click the green **Code** button
2. Make sure **HTTPS** is selected (not SSH)
3. Click the copy icon next to the URL

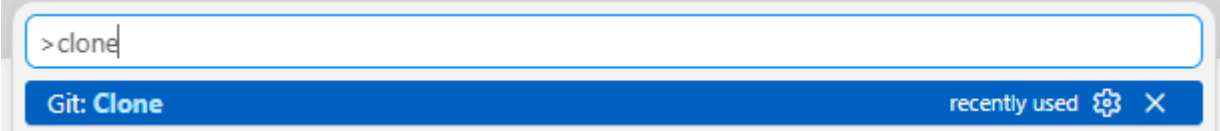


Keep this URL handy for the next step.

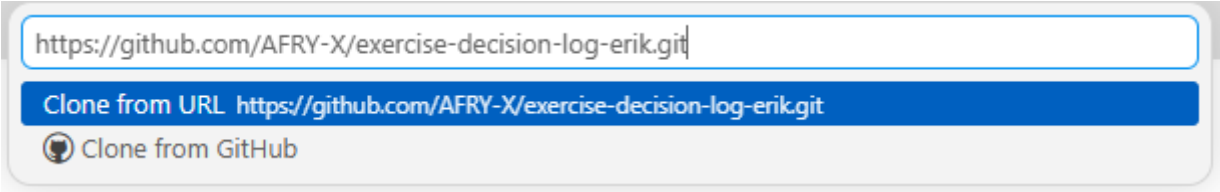
5.5.4 Step 3 - Clone the repository in VS Code

Cloning downloads the repository to your computer and links it to the remote on GitHub.

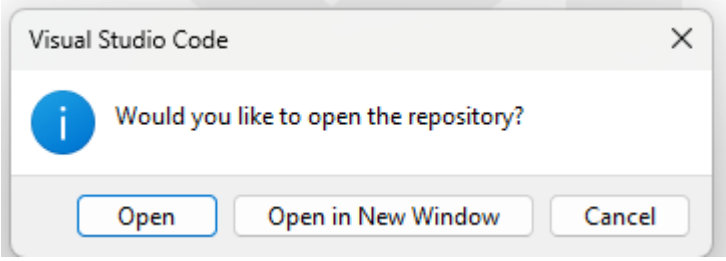
1. Open the Command Palette in VS Code with `Ctrl+Shift+P`
2. Type `clone` and select **Git: Clone**



1. Paste the URL you copied and select **Clone from URL**



1. A folder picker opens - choose where you want the project to live. A folder like `C:\Users\YourName\git\` is a good convention as it keeps all your repositories together
2. When cloning finishes, VS Code asks whether to open the repository - click **Open**



VS Code will reload with the repository folder open (it is empty).

5.5.5 Verify

Open a terminal in VS Code from the menu **Terminal > New Terminal**. In the window that appears, type:

```
git status
```

You should see:

```
On branch main Your branch is up to date with 'origin/main'. nothing to commit, working tree clean
```

If you see that, the repository is cloned, connected to GitHub, and ready to go.

You are ready to start 16.3 - Create a web app.

5.6 Vibe Coding: Create a Web App

🎬 Video

This is where the building starts. By the end of this chapter, you will have a working web app running locally on your computer - something you can open in a browser, see on screen, and continue building on in the next steps.

5.6.1 What you are building

The app is called **Decision Log** - a simple tool for recording decisions made in a project: what was decided, why, and when.

It will be a web app, which means it runs in a browser like any website. You will also make sure it looks good on a phone - the layout will adapt to smaller screens automatically. You do not need to worry about making it installable as a phone app (that is a feature called a PWA, or Progressive Web App). For our purposes, it just needs to work well in a mobile browser, and the skill handles that automatically.

5.6.2 What the skill is going to do

The `vibe-create-web-app` skill will set up the complete starting structure for your app - all the files and folders a modern web app needs. You just need to tell it the name of your app and a couple of preferences, and it does the rest.

Specifically, it will:

- Create a **React** app - React is the tool that builds the visible parts of your app (buttons, tables, forms)
- Set up **Tailwind CSS** - a styling system that makes it straightforward to make things look good without writing custom stylesheets
- Apply **AFRY's brand guidelines** (if you want it to) - the right fonts, colours, and spacing so the app looks consistent with other AFRY tools
- Start a local **development server** so you can see the app in your browser immediately

You do not need to understand any of these technologies to continue. Copilot knows how they fit together.

5.6.3 Steps

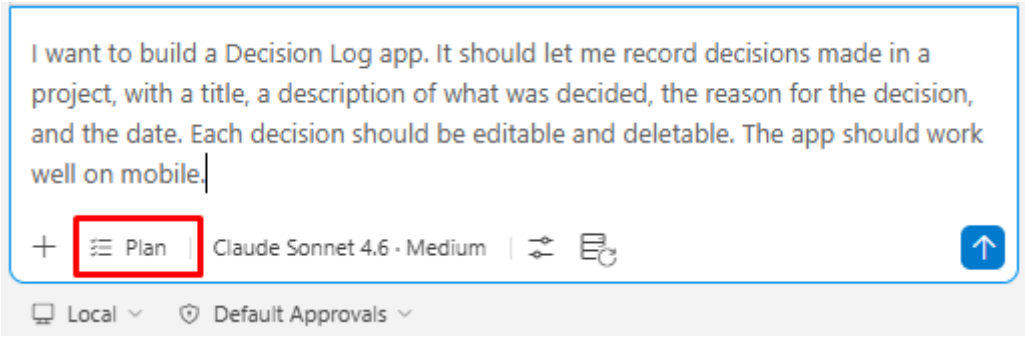
1. Open Copilot in plan mode

In VS Code, open the Copilot chat panel. Switch to **plan mode**. Select the model **Claude Sonnet 4.6** from the model picker at the top of the chat panel - this is the model that the vibe coding skills has been benchmarked against (though others work well too). Switch to **plan mode**.

In your own words, tell Copilot what you want to build. For example:

"*I want to build a Decision Log app. It should let me record decisions made in a project, with a title, a description of what was decided, the reason for the decision, and the date. Each decision should be editable and deletable. The app should work well on mobile.*"

Feel free to add more detail or change the description to match what you actually want. The more context you give, the better the plan will be.



You are in plan mode - Copilot will think and respond with a plan, but will not change any files yet. Read through it and make sure it matches what you had in mind. If something is missing or wrong, just say so and it will revise the plan.

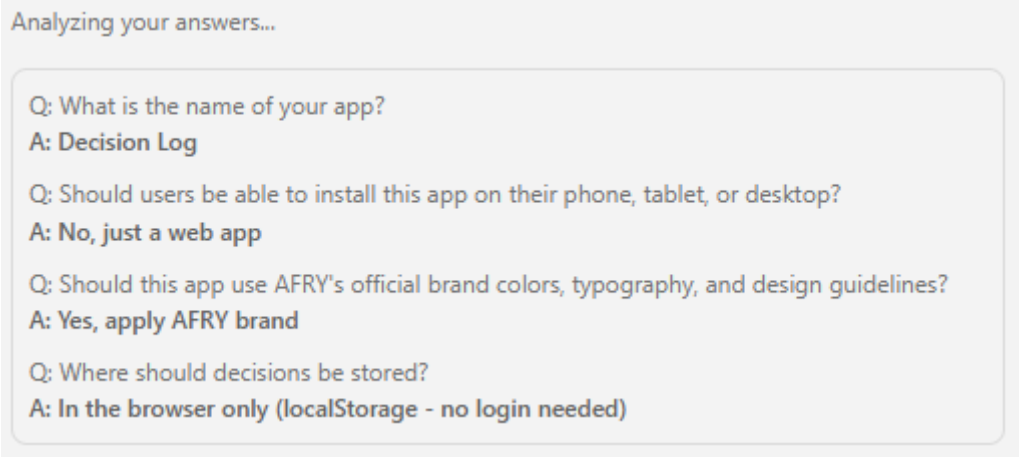
2. Answer the skill's questions

The skill will ask you some short questions before it starts:

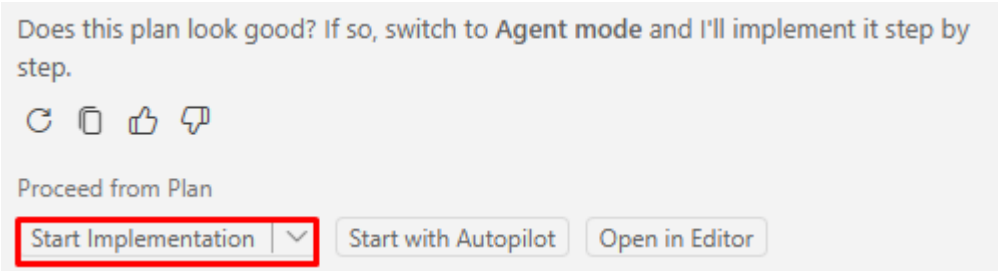
Question	What to answer
What is the name of your app?	`Decision Log`
Should users be able to install this app on their phone, tablet, or desktop?	**No, just a web app**
Should this app use AFRY's official brand colors, typography, and design guidelines?	**Yes, apply AFRY brand**

Answer each one and the skill will proceed automatically.

Note - if the assistant asks you where data should be stored, select an option for "in the browser" - we will add a real database later.



3. Start implementation



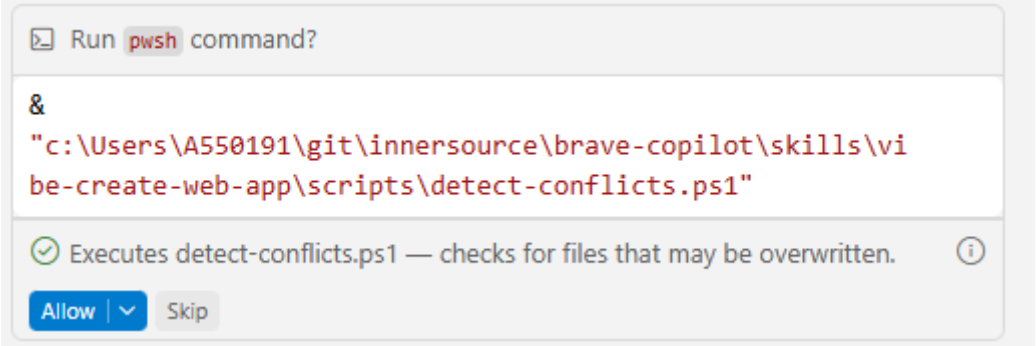
Read through the plan. You can open the plan and edit it yourself if you are not happy, or you can chat with the assistant to update it for you. When the plan looks right, click **Start implementation** (or switch to Agent yourself and type "looks good, go ahead"). Copilot will switch from planning to building.

At this point it will look for the right skill to use. Watch for it to mention **vibe-create-web-app** - that is the skill responsible for setting up the app. If it does not find it automatically, you can nudge it:

"*Please use the vibe-create-web-app skill.*"

4. Watch it build

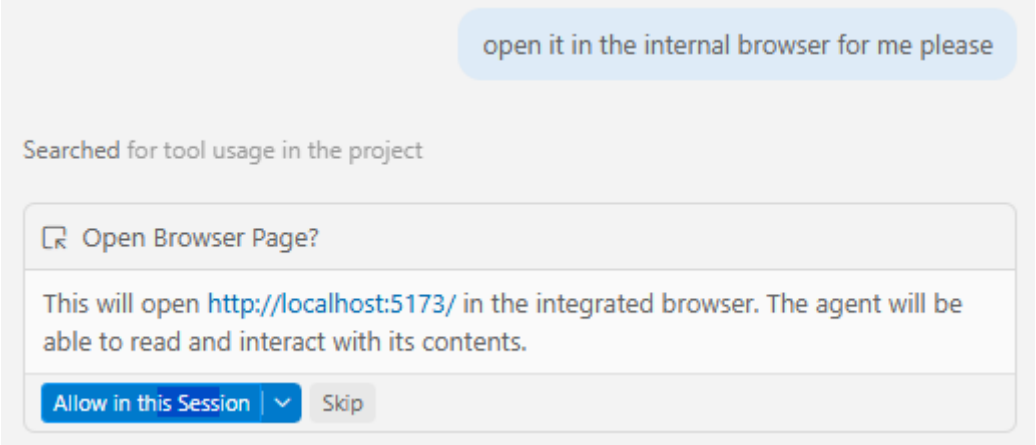
Copilot will now set up the project files and install the required packages. This takes a minute or two. When the assistant needs to run scripts, or do other things like editing configuration files, it will ask your permission.



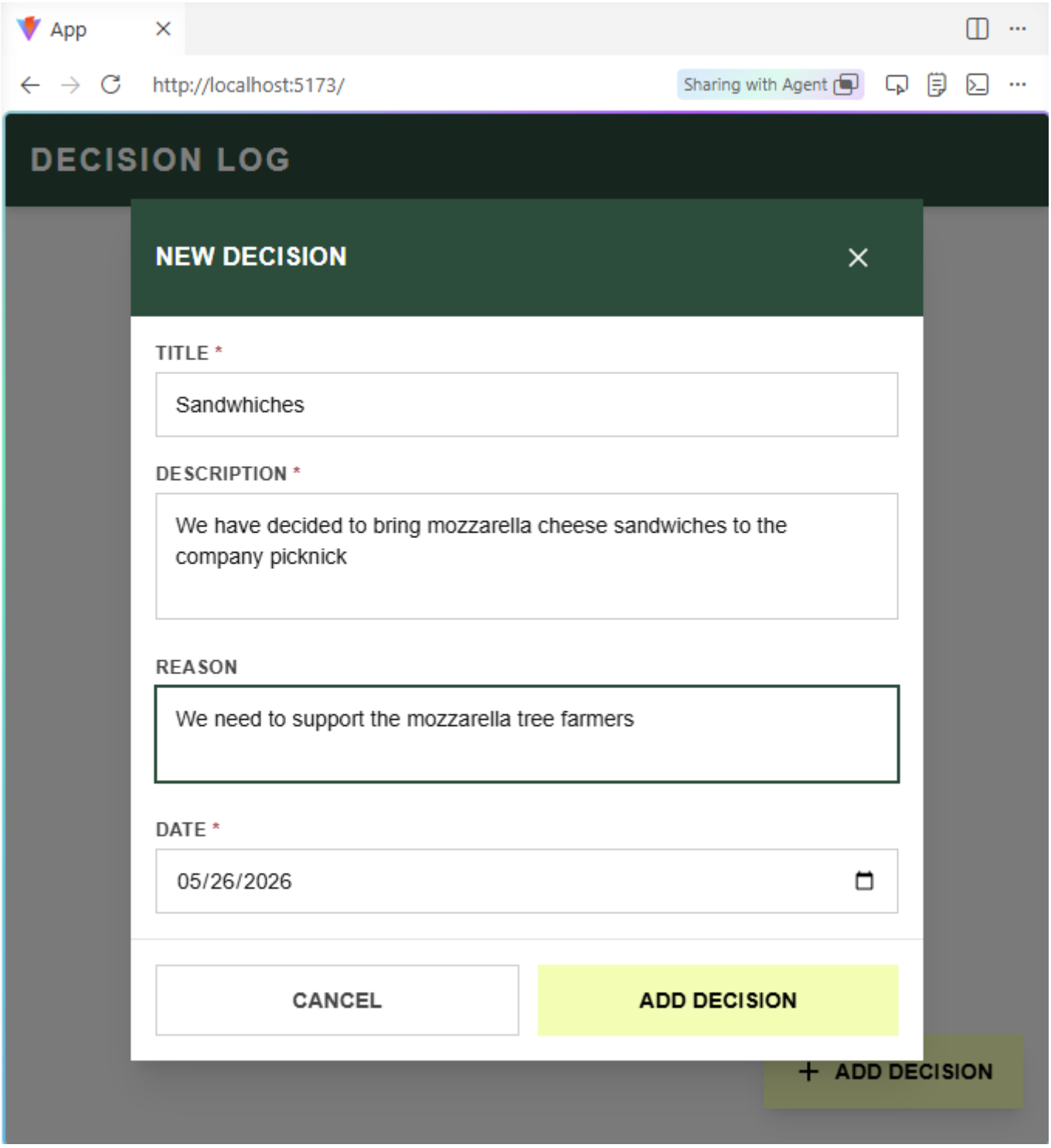
When it is done, it will tell you the app is ready.

5. Launch the app

Ask Copilot to open the app for you:



It will start the development server and open the app inside VS Code. You can also open the link in your own browser, but having the app in the internal browser allows Copilot to take screenshots and to navigate to find errors.



6. If there are errors

Even with skills, it is possible that Copilot makes mistakes. There might be errors in the configuration files, compilation problems or the app doesn't work as intended. In this case, you need to tell the agent what went wrong. You can make screenshots and paste in the chat, or ask Copilot to do so. Sometimes you must open the developer tools and give it information from the debugging tools - Copilot will guide you on how to do this.

5.6.4 What just happened

Copilot set up a complete modern web app project in your repository folder. The key files are:

What	Where
The main app component	`src/App.tsx`
Styling	Tailwind CSS, already wired up
Brand guidelines	Applied by the AFRY brand skill
Dev server config	`vite.config.ts`

Every time you save a file, the browser will update automatically - no refreshing needed.

You are ready to continue with 16.4 - Deploy to Azure.

5.7 Vibe Coding: Deploy to Azure

🎬 Video

Your app is running locally - but only on your own computer. In this chapter you will publish it to Azure so it gets a real URL that anyone can open in a browser.

Before you start, make sure you have the name of your **Azure resource group** ready. This is the container in Azure where your app will live - you set it up as part of the prerequisites. If you have not done that yet, go back to 16.2 - Prerequisites first.

5.7.1 What you are deploying to

Your app will be hosted on **Azure Static Web Apps** - a Microsoft service built specifically for frontend apps like the one you just built. It gives you:

- A public URL your colleagues can open immediately
- Automatic deployments every time you push a change to GitHub
- A free tier with no monthly cost

This last point matters: deploying your app right now costs nothing. The free tier is perfectly fine for a personal project. In a later step (adding AFRY login) you will need to upgrade to the Standard tier, which costs around €8 per month - but that is only when you are ready for it.

5.7.2 What the skill is going to do

The `vibe-deploy-app` skill handles the entire deployment process. You do not run any commands yourself. Specifically it will:

- Detect your app and confirm the deployment settings with you
- Create an **Azure Static Web App** resource in your resource group
- Set up a **GitHub Actions** **workflow** - an automated pipeline that builds and publishes your app every time you push code to GitHub
- Trigger the first deployment and wait for it to succeed

When it is done, you will have a live URL.

5.7.3 Before you start - two things to have ready

The skill will ask you two questions. Have the answers ready before you begin:

Question	What to answer
App name	Copilot will suggest a name based on your repository name - you can accept it. The format is `stapp-yourname-decision-log`
Resource group	The name of the Azure resource group you created in the prerequisites (for example `rg-abc123-01`)

5.7.4 Steps

1. Open Copilot in agent mode

In VS Code, open the Copilot chat panel. Make sure it is set to **Agent mode** (not Ask or Edit). Select **Claude Sonnet 4.6** from the model picker.

You do not need plan mode for this step - go straight to agent mode.

Tell Copilot you are ready to deploy:

2. Log in to Azure and GitHub

Copilot will check whether you are already logged in to both Azure and GitHub. If not, it will walk you through the login steps one at a time.

For **Azure**: it will open a browser window where you sign in with your AFRY account. Follow the prompts and return to VS Code when done.

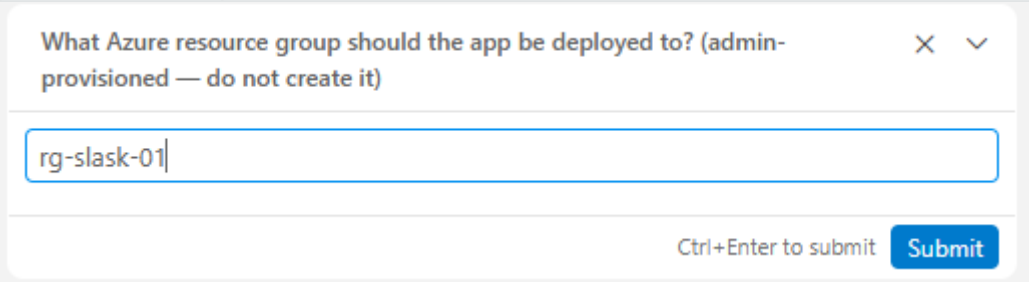
For **GitHub**: it will give you a short code and a URL. Open the URL, enter the code, and authorize the connection. Return to VS Code when done.

Once both logins are confirmed, Copilot will continue automatically.

3. Answer the skill's questions

Copilot will show you what it has detected about your app, then ask for the resource group.

Question	What to answer
Resource group	Type the name of your resource group exactly as it was given to you in the prerequisites



After you answer, Copilot will validate everything before proceeding. If something does not match - for example the resource group does not exist in the subscription - it will tell you clearly what went wrong.

4. Watch the deployment run

Copilot will now:

1. (It might initially push the code to GitHub)
2. Create the Azure Static Web App resource in your resource group
3. Commit a GitHub Actions workflow file to your repository - this means that whenever you send (push) new changes to GitHub, the deployed app will be rebuilt automatically
4. Push the code to GitHub, which triggers the first automated build and deployment
5. Wait for the workflow run to succeed

This takes a few minutes. You will see progress messages in the chat. You do not need to do anything.

5. Open your live app

When the deployment is complete, Copilot will show you the URL of your live app. Open it in a browser (or ask the agent to show it in the)

It should look exactly like the local version - the same app, now publicly accessible.

6. Check your resources log

After deployment, the skill writes a record of everything it created to a file called `docs/resources.md` in your repository. Open it and you will see a table like this:

Field	Value
App name	`stapp-exercise-decision-log-erik`
Resource group	`rg-slask-01`
Subscription	`sub-AFRY-VIBE-01-dev`
Live URL	https://kind-beach-07c432d03.7.azurestaticapps.net
GitHub repo	https://github.com/AFRY-X/exercise-decision-log-erik
GitHub Actions workflow	`.github/workflows/deploy.yml`
GitHub secret	`AZURE_STATIC_WEB_APPS_API_TOKEN`

Your values will be different - the app name, resource group, and URL are specific to your deployment. Keep this file - each subsequent step (authentication, database, AI) will add its own section to it. It becomes a useful reference for what has been set up and where to find things.

5.7.5 What just happened

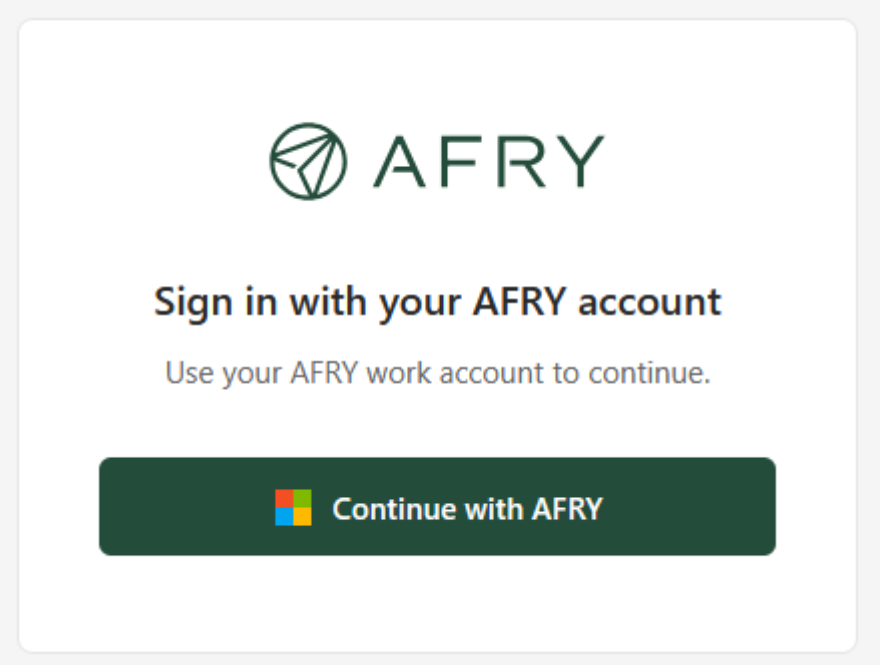
Your app is now continuously deployed. Every time you push a change to GitHub, the workflow runs automatically and your live app updates. You never need to manually redeploy.

What	Where
Azure resource	Static Web App in your resource group
Deployment pipeline	`.github/workflows/` in your repository
Live URL	Shown by Copilot after deployment, also visible in the Azure portal
Cost	€0/month (Free tier)

You are ready to continue with 16.5 - Add Authentication.

5.8 Vibe Coding: Add Authentication

🎬 Video



Right now your app is publicly accessible to anyone with the URL. In this chapter you will lock it down so only AFRY employees can sign in - using the same Microsoft account they use for everything else at AFRY.

This step uses an **App Registration** - an entry in Microsoft Entra (formerly Azure Active Directory) that identifies your app and grants it permission to ask "who is this person?" when someone visits. The app registration was created for you as part of the prerequisites, along with your resource group. If you have not done that yet, go back to 16.2 - Prerequisites first and make sure you have both the resource group name and the app registration name.

5.8.1 What you are adding

When you are done, anyone visiting your app will see an AFRY-branded sign-in page. They click sign in, go through the familiar Microsoft login, and land on the app. Unauthenticated visitors are always redirected - they cannot access any page without signing in first.

Local development is not affected. Authentication is enforced by the Azure Static Web Apps platform and only applies to the deployed app. When you run the app locally with `npm run dev`, there is no login - you can access everything directly. This is intentional and convenient: you do not need to sign in every time you make a change during development.

5.8.2 Cost

Adding authentication requires upgrading your Static Web App from the free tier to the **Standard tier**, which costs around **$9 USD per month** (billed by Azure on your subscription). This is the first step in the trail that carries a real monthly cost.

If you are not ready to accept this cost, you can skip this step and continue with the rest of the trail. Authentication can always be added later.

5.8.3 What the skill is going to do

The `vibe-add-auth-for-swa` skill handles the full setup. You do not run any commands yourself. Specifically it will:

- Upgrade your Static Web App to Standard tier
- Connect your app to the app registration in Microsoft Entra
- Configure the sign-in redirect so Microsoft knows to send users back to your app after login
- Add a route rule that blocks all unauthenticated access
- Scaffold a sign-in page in your app with AFRY branding
- Add a user header to the app so the signed-in name can be displayed

---

5.8.4 Before you start - one thing to have ready

The skill will ask you one question:

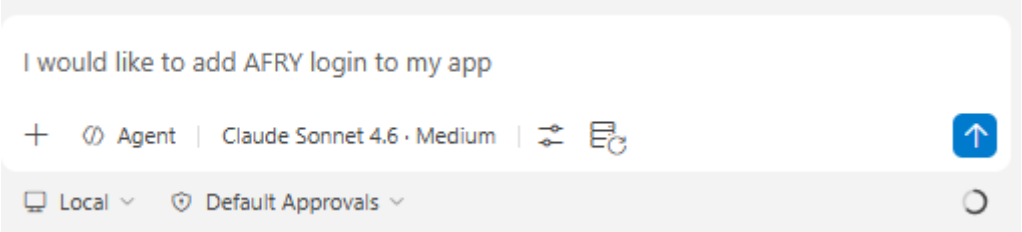
Question	What to answer
App registration name	The name you received from AFRY IT when your resource group was created (for example `apps-dev-rg-ABC123-01-SPN`)

5.8.5 Steps

1. Open Copilot in agent mode

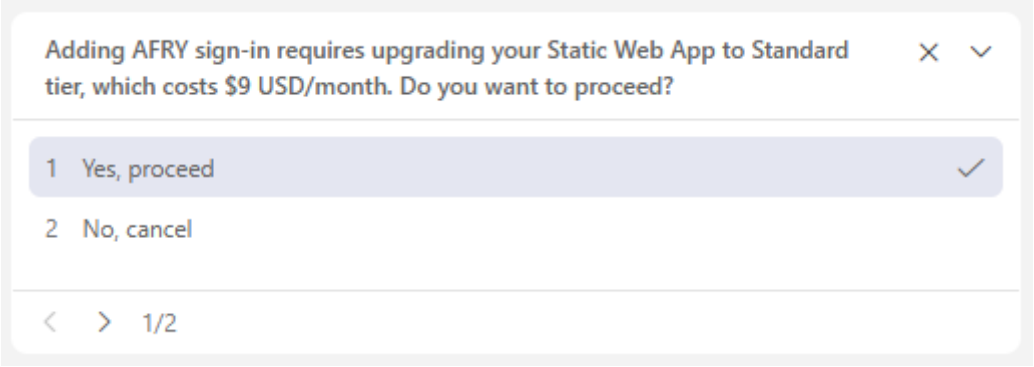
In VS Code, open the Copilot chat panel. Make sure it is set to **Agent mode** and select **Claude Sonnet 4.6** from the model picker.

Tell Copilot you want to add authentication:

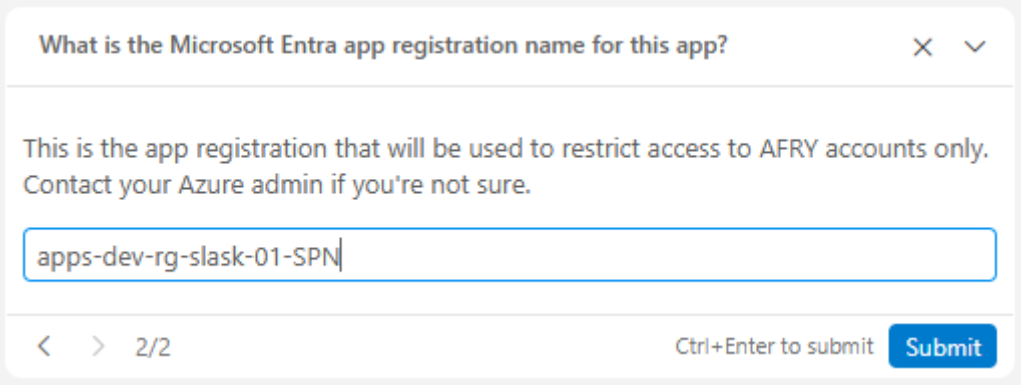


2. Answer the skill's question

Copilot will detect your deployed app automatically, then ask if you are ok with the increased costs, and then for your app registration name.



and



After you answer, the skill will validate everything before making any changes. If the app registration name is wrong or cannot be found, it will tell you clearly.

4. Watch it configure

Copilot will now:

1. Upgrade your Static Web App to Standard tier
2. Connect your app registration to the SWA auth settings
3. Update the app registration redirect URI in Microsoft Entra
4. Add a `staticwebapp.config.json` with route protection rules
5. Add a login page and logout button to your app
6. Commit and push the changes, triggering a new deployment

This takes a few minutes. You do not need to do anything.

5. Test the sign-in flow

When everything is done, open your live app URL. You should be redirected to the AFRY sign-in page. Sign in with your AFRY Microsoft account and you will land on the app.

5.8.6 What just happened

Your app is now protected. The authentication is handled by Microsoft Entra and enforced by the Static Web App platform - your app code does not need to manage tokens or sessions directly.

What	Where
Route protection	`staticwebapp.config.json` in your repository
Sign-in page	`src/pages/Login.tsx` (or `login.html` for plain HTML apps)
Auth provider	Microsoft Entra, via the app registration
Signed-in user	Available to the app from `/.auth/me`
Cost	~$9 USD/month (Standard tier)

You are ready to continue with 16.6 - Store Data.

5.9 Vibe Coding: Store Data

🎬 Video

Right now your app is a beautiful empty shell. The decisions you enter will only be visible to you because they are stored in your browser's database. In this chapter you will fix that by adding a backend, a shared database, and a proper Decision entity with full create, read, update, and delete support.

When you are done, your decisions will be saved to a real database in Azure and will still be there when you come back tomorrow, on a different device, or after sharing the link with a colleague.

5.9.1 What you are adding

Three things get added in sequence, each building on the last:

What	What it does
Backend	A set of server-side functions that your app can call to read and write data securely
Remote store	A Cosmos DB database in Azure where your data is actually saved
Decision entity	The full data model for a decision - the TypeScript type, the database operations, and the API endpoints

You will trigger each one with a single message to Copilot.

5.9.2 Cost

The remote store uses **Azure Cosmos DB**. As covered in the prerequisites, the shared AFRY subscription does not have a free tier available, so your database will run on serverless billing. For a small personal app, this typically costs well under €1 per month.

5.9.3 Step 1. Ask Copilot to add a store

Tell Copilot:

"*I want to add a shared store so my data persists.*"

Copilot will detect that a backend does not exist yet, scaffold one automatically, then provision the Cosmos DB account and wire the connection into the backend. It reads your `docs/resources.md` to find the resource group - you do not need to answer any questions.

This takes about 2-3 minutes in total. You will see progress in the chat. You do not need to do anything.

5.9.4 Step 2. Add the Decision entity

With the store in place, you can now add the actual data type your app needs.

Ask Copilot to add the Decision entity

Tell Copilot what you want to store. Be descriptive - the more context you give, the better the generated type will be:

"*I need to store decisions. Each decision has a title, a description of what was decided, the reason for the decision, and a date. Decisions should be editable and deletable.*"

Confirm the inferred interface

Before writing any code, the `vibe-add-entity` skill will show you the TypeScript interface it has inferred from your description. It will look something like this:

```
interface Decision { id: string; userId: string; title: string; description: string; reason: string; date: string; createdAt: string; updatedAt: string; _v: number; _type: string; }
```

Read through it and make sure the fields match what you had in mind. If something is missing or named wrong, say so and it will revise. When you are happy with it, confirm and the skill will generate all the code.

Watch it generate

Copilot will now create:

- The TypeScript type definition
- A repository with `create`, `read`, `update`, and `delete` operations
- An Azure Functions endpoint your app can call
- The UI components to list, add, edit, and delete decisions

When it is done, the app is fully functional end-to-end.

5.9.5 Step 3. Push to GitHub

The new code is on your machine but not yet live. Push to GitHub to trigger the deployment:

"*The entity looks good - can you push to GitHub so the changes go live?*"

Copilot will commit any uncommitted changes and push to `main`. The GitHub Actions workflow will run and deploy the updated app. When the workflow finishes, your live app will have full Decision Log functionality.

5.9.6 Running locally

Now that the app has a backend, running it locally requires two terminals:

Terminal	Command	What it does
1	`npm run dev`	Vite dev server (your app on port 5173)
2	`npm run dev:api`	Azure Functions host (API on port 7071)

The Vite dev server is configured to proxy any `/api/*` requests to the Functions host, so everything works together automatically.

5.9.7 What just happened

Your app now has a complete data layer. Decisions are stored in Cosmos DB in Azure and are tied to the signed-in user - each person sees only their own decisions.

What	Where
Backend functions	`api/src/` in your repository
Database	Cosmos DB account in your resource group
Decision type	`types/decision.ts`
Decision repository	`api/src/repositories/decisionRepository.ts`
API endpoint	`/api/decisions`
Cost	~€0-1/month (Cosmos DB serverless)

You are ready to continue with 16.7 - Talk to an LLM.

5.10 Vibe Coding: Talk to an LLM

🎬 Video

This is the final step of the trail. You are going to connect your app to an AI language model running in Azure - the same kind of model that powers GitHub Copilot and ChatGPT. Once it is wired up, you can ask Copilot to add any AI feature you can imagine: summarise your decisions, suggest what you might have missed, spot patterns across a list, answer questions about your data.

The backend you added in the previous chapter is already in place, so this step is mostly about provisioning the Azure OpenAI resource and choosing which model to deploy.

5.10.1 What you are adding

What	What it does
Azure OpenAI resource	A dedicated AI service in your resource group
Model deployment	The specific language model (for example GPT-4.1) that your app will call
Chat API endpoint	A `/api/chat` function in your backend that the frontend can talk to

---

5.10.2 Cost

Azure OpenAI charges **per token** - meaning per word (roughly) sent to and received from the model. There is no fixed monthly fee for the resource itself, but every call your app makes to the model is billed.

As a rough guide, 100 typical conversations with GPT-4.1 cost around €0.20. Costs scale with usage - if nobody is using the app, the cost is zero. You can monitor your spend in the Azure portal exactly as described in the prerequisites.

Before it creates anything, Copilot will ask you to confirm you accept the per-token cost.

5.10.3 What the skill is going to do

The `vibe-add-llm` skill handles the full setup. Specifically it will:

- Create an Azure OpenAI resource in your resource group
- Show you the list of available models and let you pick one
- Deploy your chosen model
- Wire the endpoint and authentication into your backend
- Add a `/api/chat` starter function so your app can call the model

5.10.4 Before you start - one thing to think about

You will be asked to pick a model from a list. For this trail, **GPT-4.1** is a good choice - it is capable, cost-effective, and generally available. If you see a model marked `[Preview]`, be aware it may require additional access approval from Microsoft.

5.10.5 Steps

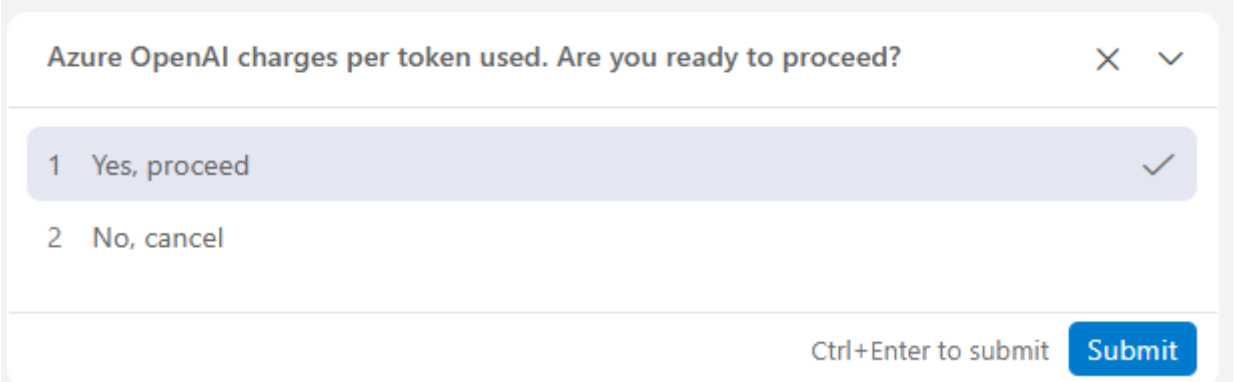
1. Open Copilot in agent mode

In VS Code, open the Copilot chat panel. Make sure it is set to **Agent mode** and select **Claude Sonnet 4.6** from the model picker.

Tell Copilot you want to add AI:

"*I want to use a LLM to check for spelling errors in the add decision dialog*"

2. Confirm the cost



Copilot will warn you about per-token billing before doing anything. Read the message and confirm you want to proceed.

3. Choose a model

Copilot will fetch the list of models available in your region and show you a multi-select picker. Select the model you want to deploy. **GPT-4.1** is recommended for this trail.

If you see `[Preview]` next to a model, it may require prior access approval from Microsoft. Stick with `[GA]` (generally available) models to avoid any delays.

4. Watch it provision

Copilot will now:

1. Create the Azure OpenAI resource (about 30 seconds)
2. Deploy your chosen model (1-2 minutes per model)
3. Grant your developer identity access so you can test locally
4. Add the `/api/chat` function to your backend
5. Wire the endpoint into your app configuration

Local development and role assignment. Step 3 above tries to grant your signed-in Azure identity the `Cognitive Services OpenAI User` role on the new resource. This requires `Owner` or `User Access Administrator` permissions on the resource group. If you do not have those permissions, the role assignment will fail silently and **local AI calls will not work** - you will get authentication errors when running `npm run dev:api`. The deployed app is not affected because it uses an API key stored in Azure app settings. If you want local dev to work, ask your Azure admin to grant you the `Cognitive Services OpenAI User` role on the Azure OpenAI resource after it is created.

5. Ask Copilot to add an AI feature

With the model wired up, you can now ask Copilot to add any AI-powered feature to your Decision Log. For example:

"*Add a button to each decision that asks the AI to suggest what risks or consequences might have been overlooked.*"

Or:

"*Add a summary panel that uses the AI to give me a brief overview of all my decisions.*"

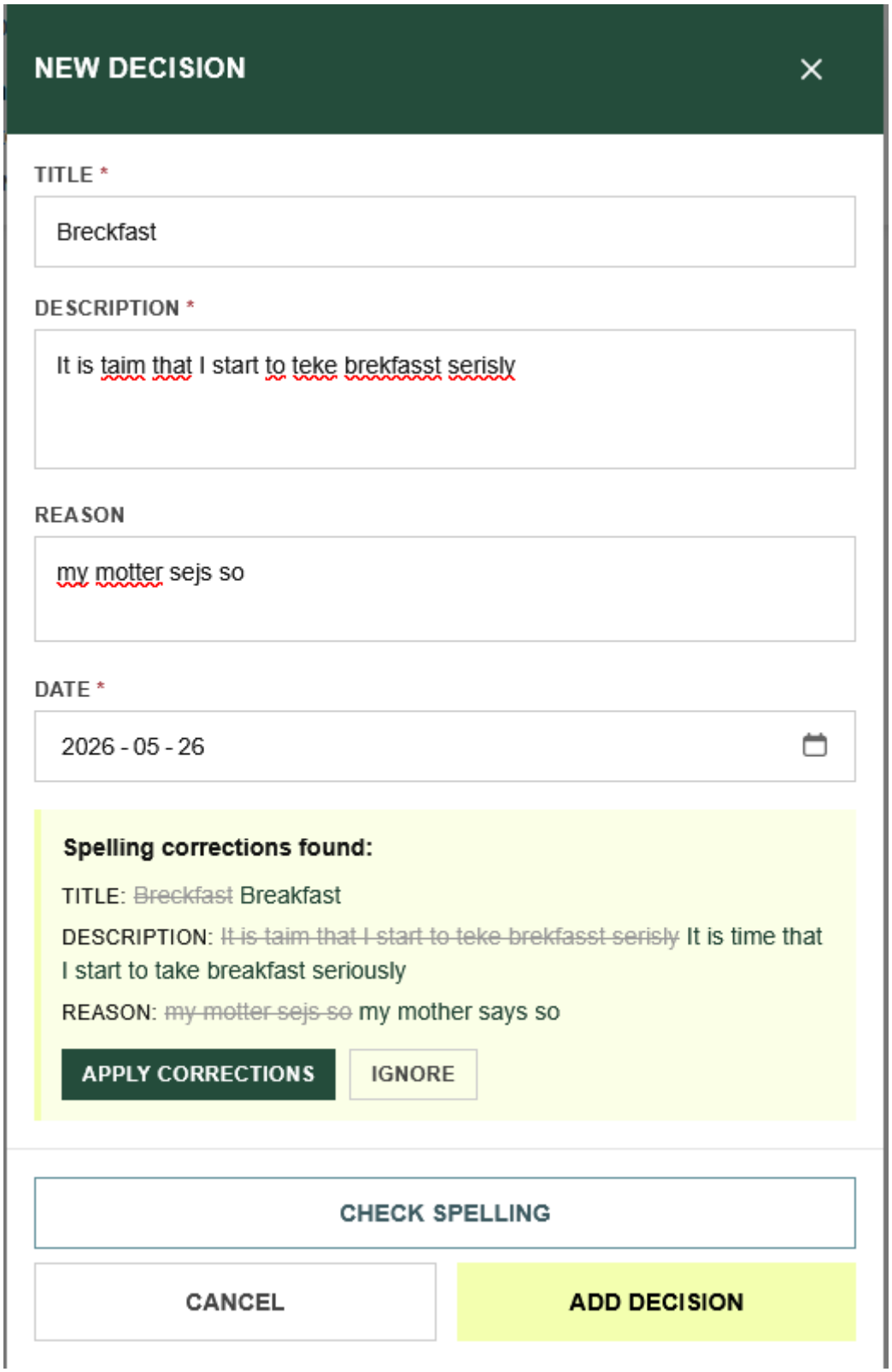
Describe what you want in plain language - Copilot knows how to call the `/api/chat` endpoint and wire it into the UI.

6. Push to GitHub

When you are happy with the result, push to deploy:

"*Looks good - please push to GitHub.*"

The GitHub Actions workflow will run and your live app will have AI features.


NEW DECISION
TITLE *
Breckfast
DESCRIPTION *
It is taim that I start to teke brekfasst serisly
REASON
my motter sejs so
DATE *
2026 - 05 - 26
Spelling corrections found:
TITLE: Breckfast Breakfast
DESCRIPTION: It is taim that I start to teke brekfasst serisly It is time that I start to take breakfast seriously
REASON: my motter sejs so my mother says so
APPLY CORRECTIONS
IGNORE
CHECK SPELLING
CANCEL
ADD DECISION

5.10.6 What just happened

Your app is now connected to a real language model. The AI runs in Azure, in your own resource group, billed to your subscription. The model never sees data it is not sent - privacy is controlled entirely by what your app chooses to include in each request.

What	Where
Azure OpenAI resource	Your resource group in the Azure portal
Model deployment	Visible under "Model deployments" in Azure OpenAI Studio
Chat endpoint	`api/src/functions/chat.ts`
Cost	Per token - roughly €0.20 per 100 conversations with GPT-4.1

5.10.7 Congratulations

You have completed the vibe coding trail. Starting from a blank repository, you have built and deployed a real web application with authentication, a persistent database, and AI features - without writing a single line of code from scratch.

What you built is a starting point, not a finished product. Every part of it can be extended, restyled, or replaced. The skills you used here work just as well for your next app idea.

6. Glossary

Quick reference for terms used across the vibe coding trail and the GitHub Copilot series.

6.1 Agent mode

The mode in GitHub Copilot where the AI acts autonomously - it reads files, writes code, runs commands, and fixes errors, continuing until the task is done. You describe the goal; the agent figures out the steps.

6.2 API (Application Programming Interface)

A defined way for two pieces of software to talk to each other. When your app needs to send data to a server, or get data back, it does so through an API. You do not usually need to build APIs by hand - the vibe coding skills generate them for you.

6.3 App Registration

An entry in AFRY's identity system (Microsoft Entra ID) that represents your application. It is what allows your app to use "Sign in with AFRY" - the identity system needs to know your app exists before it will issue login tokens to it. Managed by AFRY IT admins.

6.4 Authentication (Auth)

The process of verifying who a user is - typically by asking them to sign in. In the vibe coding trail, authentication is handled by Microsoft Entra ID (AFRY's identity platform), so users sign in with their existing AFRY credentials.

6.5 Azure

Microsoft's cloud platform. Azure provides the servers, storage, databases, and networking that your deployed app runs on. AFRY uses Azure as its standard cloud environment. See also: Resource group, SWA, Azure Functions, Cosmos DB.

6.6 Azure CLI (az)

A command-line tool that lets you manage Azure resources from a terminal. The command is `az`. The vibe coding skills use it to create and configure Azure services on your behalf. You install it once and log in with `az login`.

6.7 Azure AI Foundry

Microsoft's platform for building and deploying AI models on Azure. When the vibe coding trail connects your app to a language model, it uses an Azure OpenAI resource provisioned through Foundry.

6.8 Azure Functions

A way to run small pieces of server-side code in Azure without managing a full server. When your app needs a backend - for example, to talk to a database or call an AI model - Azure Functions is how the vibe coding skills add one.

6.9 Branch

A separate line of development in a Git repository. Creating a branch lets you work on a new feature or change without touching the main version of the project. When you are done, you merge the branch back in.

6.10 CI/CD (Continuous Integration / Continuous Deployment)

An automated pipeline that builds and deploys your app every time you push code to GitHub. Once set up, you never manually deploy again - every push to the main branch triggers a fresh deployment automatically.

6.11 Commit

A saved snapshot of your project at a specific point in time, with a short message describing what changed. Commits are the building blocks of Git history.

6.12 Cosmos DB

Microsoft Azure's cloud database service. In the vibe coding trail, it is the database that stores your app's shared data. Managed entirely by Azure - you never set up or maintain a server yourself.

6.13 Entra ID (formerly Azure Active Directory / Azure AD)

AFRY's identity platform, managed by Microsoft. It is the system behind "Sign in with your AFRY account". When you add authentication to your app, Entra ID is what checks credentials and issues login tokens.

6.14 Git

The version control tool installed on your computer. It tracks every change you make to your project files, lets you roll back to any earlier state, and coordinates work when multiple people contribute to the same project.

6.15 GitHub

The cloud platform that hosts Git repositories. GitHub adds collaboration features (pull requests, code review, issue tracking) and automation (GitHub Actions). AFRY's code lives in GitHub under several organisations.

6.16 GitHub Actions

An automation system built into GitHub. It runs workflows - scripts that build, test, and deploy your app - triggered automatically whenever you push code. The vibe coding deploy skill sets up a GitHub Actions workflow so your app redeploys on every push.

6.17 GitHub CLI (gh)

A command-line tool for automating GitHub platform actions from a terminal. The command is `gh`. Used by the vibe coding deploy skill to create repositories, set secrets, and trigger deployments. Install it once and log in with `gh auth login`.

6.18 GitHub Enterprise Managed Users (EMU)

AFRY's GitHub setup. Your GitHub account is created and managed by AFRY through Entra ID - not a personal account. EMU accounts are scoped to AFRY's enterprise and cannot access or publish content outside it.

6.19 LLM (Large Language Model)

The type of AI model that powers GitHub Copilot, ChatGPT, and similar tools. An LLM can understand text and generate text - explaining things, writing code, summarising documents, answering questions. In the vibe coding trail, the final step connects your app to an LLM via Azure OpenAI.

6.20 MCP (Model Context Protocol)

A protocol that lets Copilot call external tools and services during a conversation. An MCP server exposes a set of tools - for example, looking up a colleague's profile, uploading a file, or querying a database. Once configured, Copilot can use those tools automatically in Agent mode.

6.21 MCP tools

See MCP (Model Context Protocol).

6.22 Node.js

A runtime that lets JavaScript run outside a browser - on your computer or on a server. Many tools in the vibe coding stack (Vite, npm packages, Azure Functions) require Node.js to be installed.

6.23 npm

The package manager that comes with Node.js. Used to install the libraries your app depends on (`npm install`) and to run development scripts (`npm run dev`).

6.24 Pull request (PR)

A request to merge changes from one branch into another in a GitHub repository. Pull requests are where code review happens - team members can comment, suggest changes, and approve before the code is merged.

6.25 Python

A programming language widely used for scripting, data analysis, and automation. Some Copilot tools and MCP servers require Python to be installed.

6.26 React

The JavaScript library used to build the user interface of your vibe coding app. You do not write React code manually - the `vibe-create-web-app` skill generates it for you. React makes it easy to build interactive web pages.

6.27 Repository (repo)

A project folder managed by Git, with its full change history stored alongside the files. Your vibe coding project lives in a repository hosted on GitHub.

6.28 Resource

A single Azure service - for example, a Static Web App, a database, a Functions app, or a storage account. Resources live inside a resource group.

6.29 Resource group

A container in Azure that groups related resources together. All the Azure services for a single project - the web app, the database, the functions - typically live in the same resource group. You need a resource group before you can deploy anything to Azure. Request one from AFRY IT admins.

6.30 Skill

A markdown file that gives Copilot a set of specialised instructions for a specific task. Skills are on-demand - you invoke one by mentioning its name in a prompt. The vibe coding skills (`vibe-create-web-app`, `vibe-deploy-app`, etc.) are examples. They live in the Brave Copilot repository.

6.31 SWA (Azure Static Web App)

The Azure service that hosts the front end of your vibe coding app. It serves your web pages from a global CDN (content delivery network), meaning pages load fast regardless of where your users are. It also has built-in support for authentication and for linking to an Azure Functions backend.

6.32 Tailwind CSS

A toolkit for styling web pages, used by the `vibe-create-web-app` skill. Instead of writing stylesheets, you add short class names to your HTML elements and Tailwind handles the visual result.

6.33 Terminal

A text-based window where you type commands and see output. In VS Code, open it with `Ctrl+` . The vibe coding trail uses the terminal to run install and start commands, and Copilot uses it to run Git and Azure CLI commands on your behalf.

6.34 Token (AI)

The unit of text that AI models process. Roughly, a token is about three-quarters of a word. Models charge by token - both input (what you send) and output (what they generate). Longer conversations and more complex tasks use more tokens.

6.35 Vibe coding

A way of building software by describing what you want in plain language and letting an AI assistant do the coding. No prior programming experience required - the skill is in describing the outcome clearly.

6.36 Vite

A fast development server and build tool for web apps. When you run `npm run dev`, Vite starts a local server so you can preview your app in a browser while you build it. It also compiles and bundles your app for deployment.

7. Downloads

Download the full training material as a PDF document or as a ZIP archive of the raw sources.

DOWNLOAD PDF **DOWNLOAD ZIP (MARKDOWN SOURCES)**

The **PDF** is a single-document export of the entire site, suitable for offline reading or printing.

The **ZIP** contains all original Markdown files and images, useful if you want to work with the content offline or import it elsewhere.