Context workflows

Give each task the source files it needs

Access and cost preflight

Follow the course safety baseline.

1. Check the current GitHub docs and your organization's rules.
2. Check the repository and data sources.
3. Use a small question and synthetic data.
4. Set a usage limit for metered work.

Context selection

Choose sources a reviewer can explain:

- repository instructions;
- relevant source files and tests;
- architecture or API documents;
- approved task notes.

Do not include unapproved data merely to make a prompt more complete.

Context packet exercise

```
Task: add validation to an API endpoint

Packet:
- route file
- corresponding tests
- API contract
- repository conventions
```

Ask learners to explain why each source is needed and what is excluded.

Evaluate the result

- Does it cite or follow the selected repository files?
- Is the scope limited to the task?
- Are assumptions listed for human review?
- Do relevant tests pass?

Required access

Check Spaces and GitHub Copilot access before the lab. If either is unavailable, stop the lab. A manually assembled packet does not demonstrate Space creation, reuse, or behavior.

Agenda and time plan

Time	Topic	Trainer move
0:00	Why context fails	Diagnose a vague response
0:08	Context layers	Build a source hierarchy
0:18	Spaces	Select and share sources
0:30	Instructions and prompt files	Show repository artifacts
0:42	Memory and maintenance	Separate preview from policy
0:50	Before/after demo	Compare with a rubric
0:58	Lab handoff	Select sources for the live Space

Transition: "Choose sources that answer the task before adding more prompt text."

The context problem

Copilot cannot reliably apply information it cannot see. Extra context can create noise or expose data unnecessarily.

Common failure modes:

- the relevant contract is absent;
- stale documentation conflicts with source code;
- broad context hides the key constraint;
- local preferences are mistaken for team standards;
- restricted data is included without approval.

Check both relevance and permission when selecting context. Keep the packet focused.

A practical context hierarchy

```
Task intent and acceptance criteria
                ↓
Repository instructions and prompt workflow
                ↓
Curated Space: code, docs, issues, specifications
                ↓
Current file and explicit references
                ↓
Validated memory, when supported and approved
```

Specific task constraints override generic guidance. Clarify conflicts.

Copilot Spaces

A Space is a selected set of sources for a recurring topic or workflow.

Good candidates include:

- source repositories and key paths;
- architecture and API documentation;
- approved issues and specifications;
- team guidance for a domain;
- links whose ownership and freshness are known.

Do not use a Space to collect restricted information without approval.

Curate for a purpose

```
Space: Storefront API
├── API specification
├── product route and model
├── architecture note
├── error-handling conventions
└── repository instructions
```

Exclude unrelated services, production data, secrets, abandoned proposals, and documents with no owner to keep them current.

Trainer cue: Ask learners what they would remove first if the answer became noisy.

Sharing Spaces safely

Before sharing:

1. Name the intended audience and task.
2. Check that every source is approved for that audience.
3. Add purpose-specific instructions.
4. Assign an owner and review date.
5. Test with a known question and compare against the source files.
6. Remove or replace stale sources.

Check current access controls and sharing behavior before the demo.

Repository-level instructions

Create `.github/copilot-instructions.md` for project-wide guidance:

```
# Repository guidance

- Use TypeScript strict mode.
- Validate request input at the route boundary.
- Keep business logic in `src/services/`.
- Run `npm test` before proposing completion.
- Do not add dependencies without reviewer approval.
```

Write rules a reviewer can check. Avoid vague goals such as "produce high-quality code."

Reusable prompt files

Place team-reviewed workflows under `.github/prompts/`:

```
---
description: Review an API change against its contract
---

Read the selected route, service, tests, and API contract.
List mismatches before suggesting changes.
Do not edit files outside the selected feature.
End with the focused validation command.
```

Check the current supported filename and frontmatter behavior before delivery.

Inspect what the CLI loaded

```
/env
/instructions
```

Find the copied stale CommonJS note and its route-file scope.
Compare it with `package.json` and the current ES-module route.

Correct only that training note, restart, and inspect again.
Do not toggle personal, managed, or security instructions.

Discovery is not proof of application. Check the proposed code too.
The lab includes this work in its 30-minute instruction block.

VS Code instruction settings

The curriculum references:

```
{
  "github.copilot.chat.codeGeneration.useInstructionFiles": true
}
```

Treat settings as version-sensitive:

- check current documentation and the learner's IDE version;
- show where workspace settings are reviewed;
- do not claim an instruction loaded without observing the result;
- retain the instructions as a useful manual checklist.

Copilot Memory

Memory may let an agent retain and validate useful knowledge, depending on support and approval.

Use three questions:

1. **Source:** Where did this fact come from?
2. **Scope:** Does it apply to this user, repository, or task?
3. **Freshness:** How will it be corrected when the code changes?

Never store secrets, restricted data, or guesses. Check current availability and controls.

Demo: without grounded context

Prompt:

```
Add a low-stock report endpoint.
```

Expected discussion:

- Which endpoint?
- Which inputs and error schema?
- Where is validation normally performed?
- Which tests prove the behavior?
- Which data is safe to include?

Do not score the response by confidence or length.

Demo: with a context packet

```
Using the Storefront API specification, product route, product model, architecture
note, and repository instructions, propose the smallest low-stock endpoint change.

First list acceptance criteria and assumptions. Do not add dependencies.
End with two verification requests and cite the evidence used.
```

Check correctness and scope. Compare assumptions, source references, and the checks each response proposes.

Concrete code target

```
function parseThreshold(value) {
  if (value === undefined) return 10;
  if (!/^\d+$/.test(value)) {
    throw new Error("threshold must be a non-negative integer");
  }
  return Number(value);
}
```

Ask where the static report route must appear relative to `/:id`. The correct next action may be clarification rather than code generation.

Maintenance is part of context

Trigger	Maintenance action
API contract changes	Update Space sources and prompts
Repository convention changes	Review instruction files
Ownership changes	Assign a new maintainer
Output contradicts source	Remove stale sources and retest
Feature behavior changes	Recheck official documentation

Schedule reviews; do not wait for failures.

Lab handoff

You will:

1. keep one Storefront API for the full lab;
2. select the smallest approved source set;
3. add repository instructions and a low-stock endpoint;
4. compare one logging task across context levels;
5. store and test three reusable prompt files.

Final deliverable: A Storefront API context package with comparison results and the checks you ran.

Remember

1. Relevant context matters more than a large context packet.
2. Select Space sources for a named purpose and audience.
3. Instructions state project-wide rules; prompt files define reusable tasks.
4. Memory requires source, scope, freshness, and current policy checks.
5. Name who will keep each context source current.

Questions and lab readiness

Which source would you remove from a context packet, and why?

- Which source is authoritative?
- What must stay out of the Space?
- Who will maintain the resulting artifacts?