CRÉDIT AGRICOLE
CORPORATE & INVESTMENT BANK

Microsoft Github Copilot Learning Capsule

Advanced Prompt Engineering

July, 2025

IOS
IT & OPERATIONS SERVICES

Advanced prompt engineering: objectives of this Learning Capsule

1	Identify the three pillars of an efficient prompt	10’
2	Demonstrate how to prompt efficiently	15’
3	Answer your questions and share feedback	5’

Different prompt engineering technics with AI

Zero-Shot Learning
Give GitHub Copilot a task without any prior examples. You describe what you want in detail, assuming AI has not prior knowledge of the task

One-Shot Learning
Give an example along with your prompt

Few-Shot Learning
Provide few examples to help AI understand the pattern or style of response you are looking for

Chain-of-Thought Prompting
Ask AI to detail its thought process step-by-step

Iterative Prompting
Refine your prompt based on the output you get

Negative Prompting
Tell GitHub Copilot what to not do

Prompt Chaining
Break down a complex task into smaller prompts and then chaining the output

Hybrid Prompting
Combine different methods, like few-shot learning + chain-of-thought

The 3 pillars of an efficient prompt with GitHub Copilot

INSTRUCTION

Give a clear and precise directive on the action to be performed or the code to be generated

Example

Write a Python function to sort a list in ascending order.

CONTEXT

Keep important files open in your IDE to add context to your prompt and complete the instruction

Example

The project uses Python 3.9 and the function must be adapted to large lists.

The files we are working on as part of COPILOT

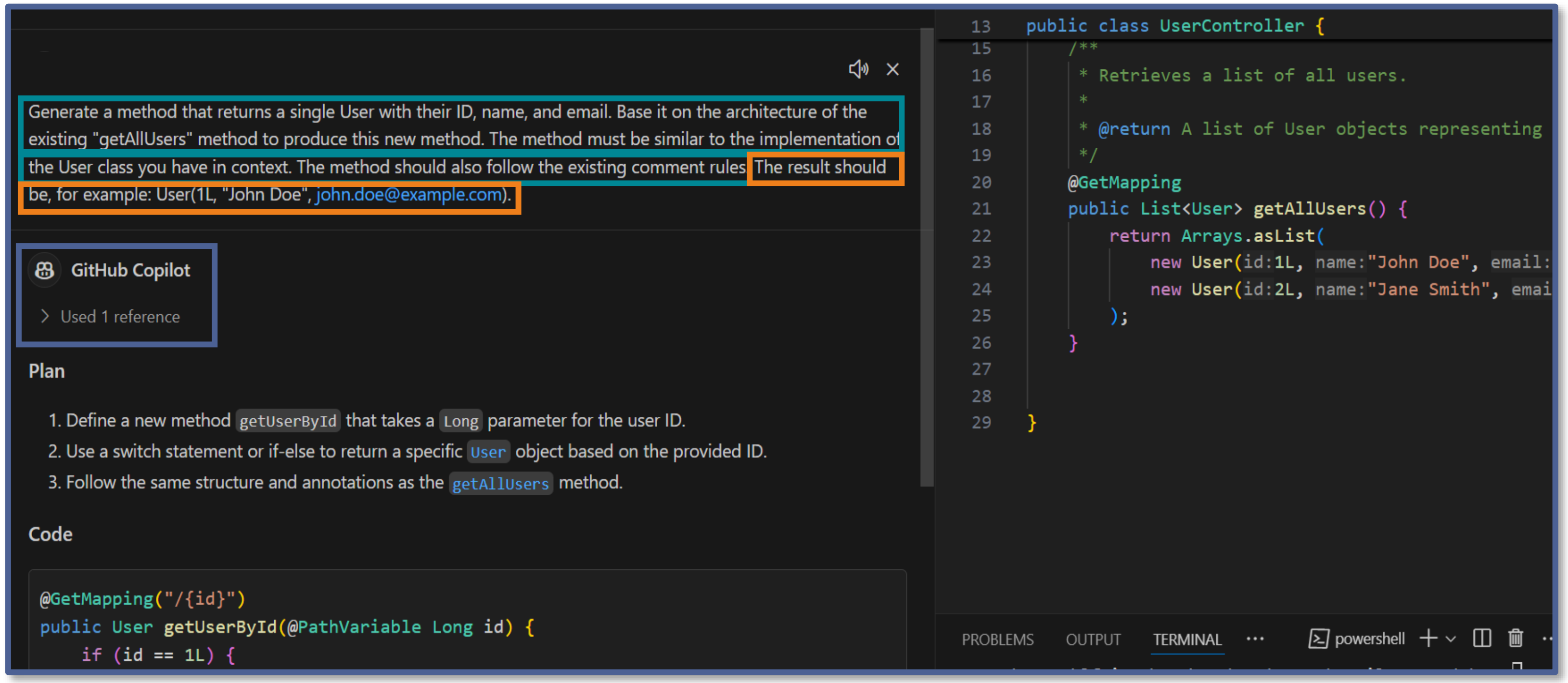
EXAMPLE

Provide an example of a precise, pre-established result to orient the result on a similar solution

Example

Draw inspiration from the sorting function you previously generated, but use an iterative approach instead of a recursive one

Applying the 3 pillars of a good prompt



Generate a method that returns a single User with their ID, name, and email. Base it on the architecture of the existing "getAllUsers" method to produce this new method. The method must be similar to the implementation of the User class you have in context. The method should also follow the existing comment rules The result should be, for example: User(1L, "John Doe", john.doe@example.com).

Plan

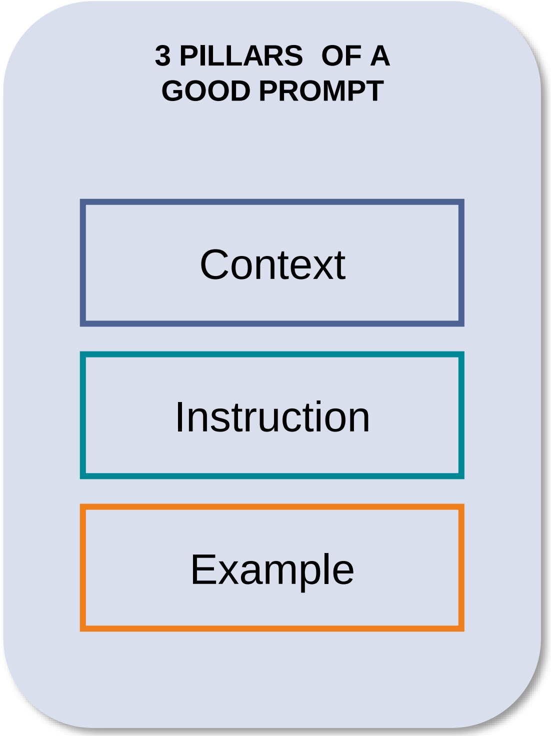
1. Define a new method `getUserById` that takes a `Long` parameter for the user ID.
2. Use a switch statement or if-else to return a specific `User` object based on the provided ID.
3. Follow the same structure and annotations as the `getAllUsers` method.

Code

```
@GetMapping("/{id}")
public User getUserById(@PathVariable Long id) {
    if (id == 1L) {
```

```
public class UserController {
    /**
     * Retrieves a list of all users.
     *
     * @return A list of User objects representing
     */
    @GetMapping
    public List<User> getAllUsers() {
        return Arrays.asList(
            new User(id:1L, name:"John Doe", email:
            new User(id:2L, name:"Jane Smith", emai
        );
    }


}
```



Generate a method that returns a single User with their ID, name and email. Base it on the architecture of the existing "getAllUsers" method to produce this new method. The method must be similar to the implementation of the User class you have in context. The method should also follow the existing comment rules. The result should be, for example: User(1L, "John Doe", john.doe@example.com)

Demonstration

Do you have any questions?

CRÉDIT AGRICOLE
CORPORATE & INVESTMENT BANK