directions lyon
emea 2023

Create your AL solutions the right way: With Design Patterns and Architecture Best Practices in mind

Patrick Schiefer, COSMO CONSULT
Tobias Fenster, 4PS

Agenda

Speaker introduction

Why consider (object-oriented) design patterns and best practices?

Patterns / best practices to discuss:
- Interfaces
- SOLID principles
- Adapters

Architecture for integrating external components

Q&A

directions lyon
emea 2023

Patrick Schiefer

DevOps Engineer
Cosmo Consult

schiefer_p

patrickschiefer@msdyn365bc.social

PatrickSchiefer

patrickschiefer.com

directions lyon
emea 2023

Tobias Fenster

Managing Director
4PS Germany

Microsoft Regional Director and dual MVP
Docker Captain

tobiasfenster
tobiasfenster@hachyderm.io
tfenster
tobiasfenster.io

Why?

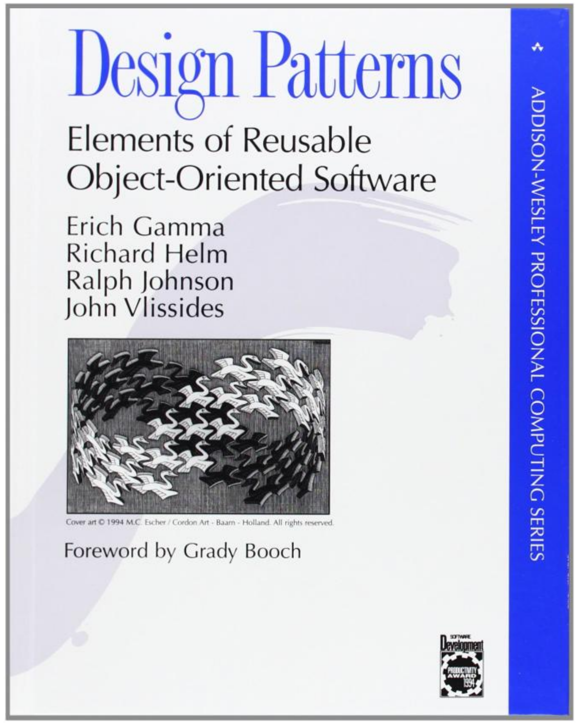
Design patterns and best practices

- Recognizable, repeatable solutions for similar problems
  - “We need a place to store the cars” → garage
  - Could also be car port, multi-storey car park, underground car park…
- Continuously learn and improve on implementing those patterns
- Have a common vocabulary for architectural and conceptual discussions
- Make maintenance and improvement of existing code easier
  - You know how it works without digging into the details

Why?

Object-oriented design patterns and best practices

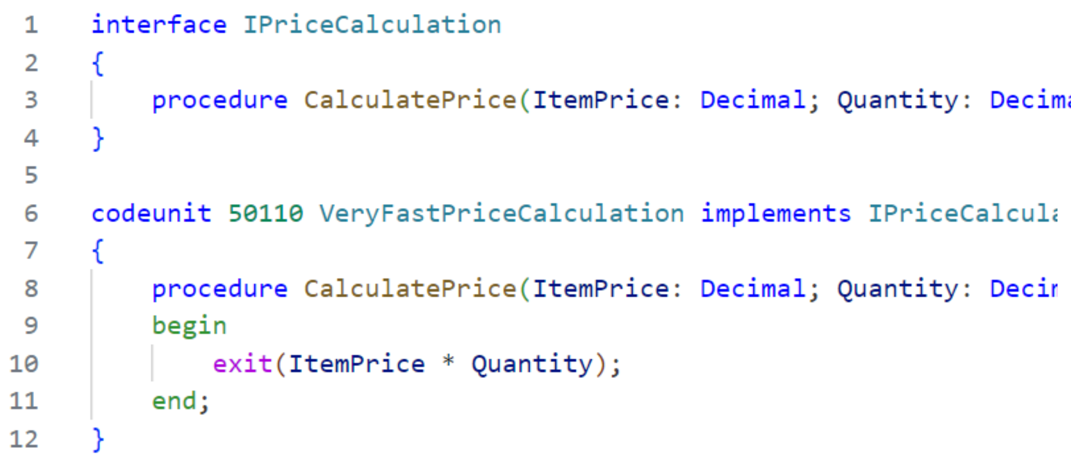
- Well established catalogue
  - Early, probably most famous: Design Patterns: Elements of Reusable Object-Oriented Software (1994) by "Gang of Four" (Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides)
- If new people know design patterns, it's probably object-oriented design patterns
- A lot of them work quite well although AL is not object-oriented


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

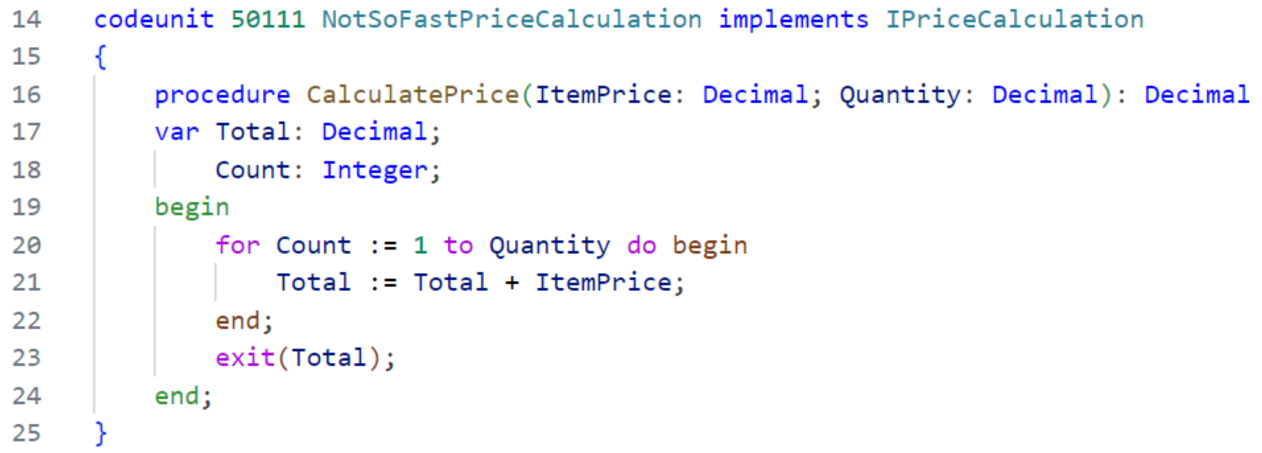
Interlude: Interfaces

What can it do and why do we need it?

- Right from the official docs: An interface in AL is similar to an interface in any other programming language; it's a **syntactical contract** that can be implemented by a non-abstract method. The interface is used to **define which capabilities must be available** for an object, while **allowing actual implementations to differ**, as long as they comply with the defined interface.



```
interface IPriceCalculation
{
    procedure CalculatePrice(ItemPrice: Decimal; Quantity: Decimal): Decimal;
}

codeunit 50110 VeryFastPriceCalculation implements IPriceCalcula
{
    procedure CalculatePrice(ItemPrice: Decimal; Quantity: Decir
    begin
        exit(ItemPrice * Quantity);
    end;
}
```



```
codeunit 50111 NotSoFastPriceCalculation implements IPriceCalculation
{
    procedure CalculatePrice(ItemPrice: Decimal; Quantity: Decimal): Decimal
    var Total: Decimal;
        Count: Integer;
    begin
        for Count := 1 to Quantity do begin
            Total := Total + ItemPrice;
        end;
        exit(Total);
    end;
}
```

SOLID

What’s the idea?

- First described by “Uncle Bob” Martin in 2000

- A collection of principles to help with software design
  - Not rules, not laws, not magic bullets
  - Same as design patterns: help to discuss, understand, follow good ideas; not laws or magic bullets → Buggy SOLID code is still buggy code

- **S**ingle-responsibility
- **O**pen-closed
- **L**iskov substitution
- **I**nterface segregation
- **D**ependency inversion

Single-responsibility principle

What's the idea?

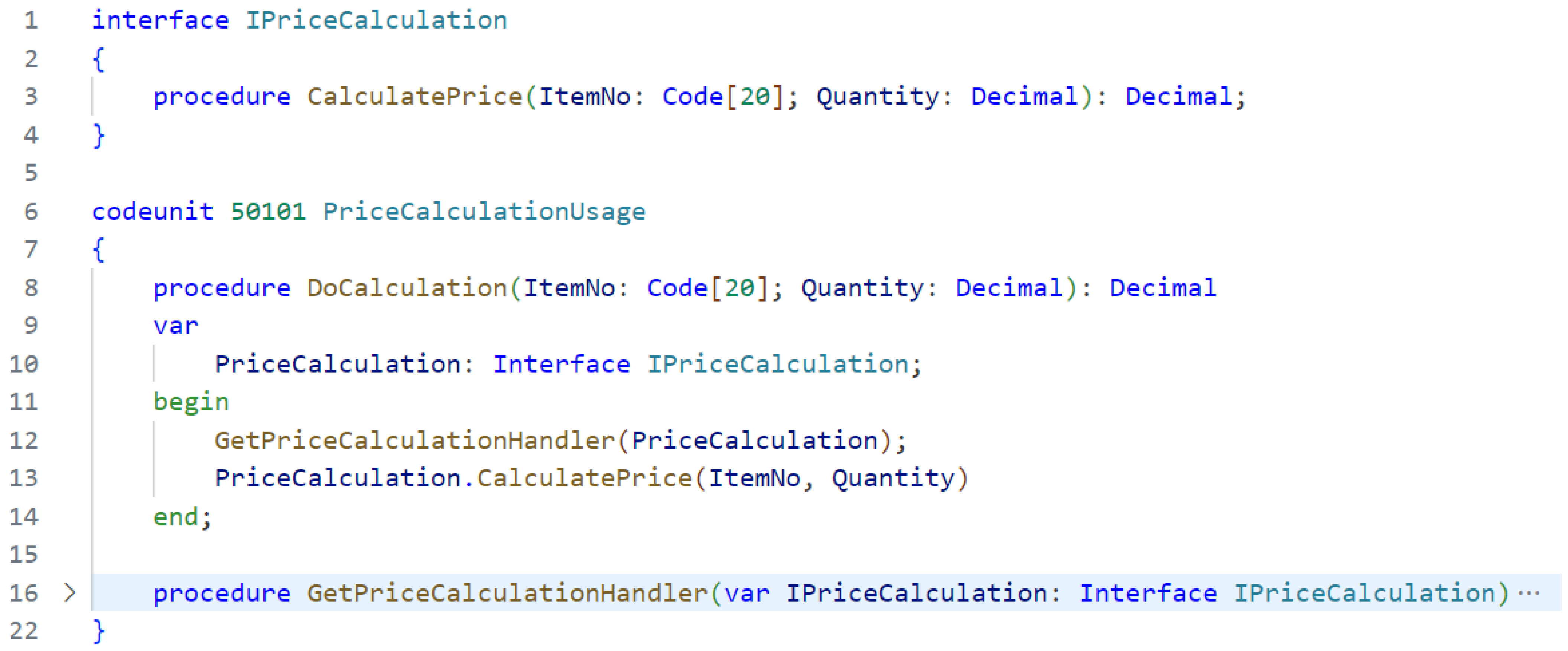
"There should never be more than one reason for an object to change" or every object has exactly on responsibility

```
codeunit 50100 PriceCalculation
{
    procedure CalculatePrice(ItemNo: Code[20]; Quantity: Decimal): Decimal
    begin
        ...
    end;
}

codeunit 50101 CustomerRecordFunctions
{
    procedure DeleteCustomerRecord(CustomerNo: Code[20])
    begin
        ...
    end;
}
```

Open-closed principle

What’s the idea?

“Objects should be open for extension, but closed for modification”

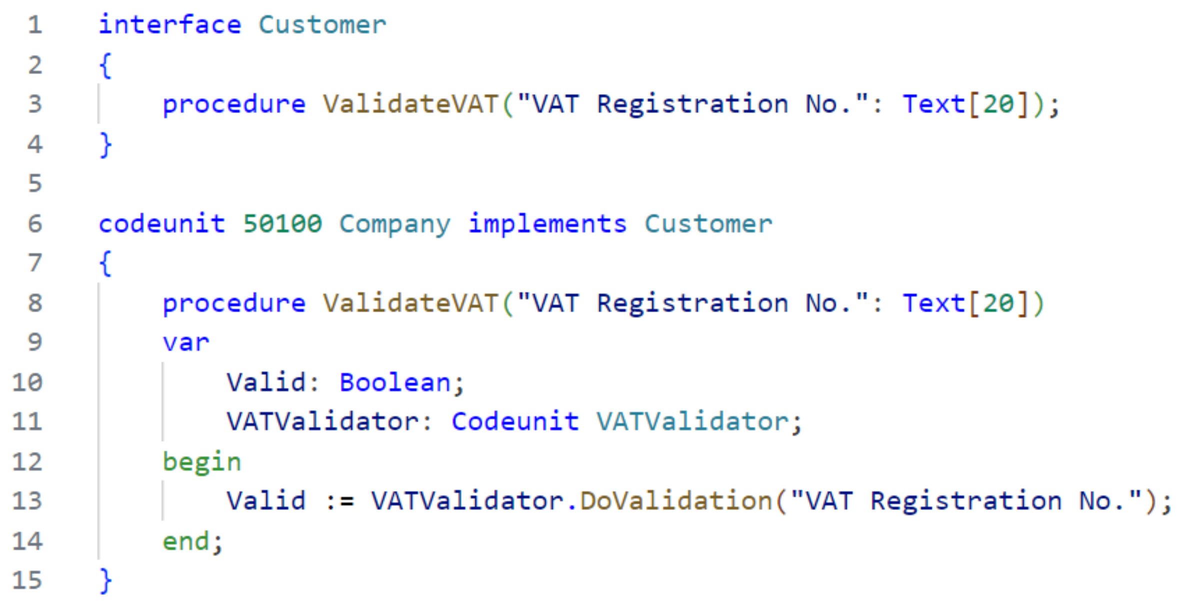


```
interface IPriceCalculation
{
    procedure CalculatePrice(ItemNo: Code[20]; Quantity: Decimal): Decimal;
}

codeunit 50101 PriceCalculationUsage
{
    procedure DoCalculation(ItemNo: Code[20]; Quantity: Decimal): Decimal
    var
        PriceCalculation: Interface IPriceCalculation;
    begin
        GetPriceCalculationHandler(PriceCalculation);
        PriceCalculation.CalculatePrice(ItemNo, Quantity)
    end;

    procedure GetPriceCalculationHandler(var IPriceCalculation: Interface IPriceCalculation)···
}
```

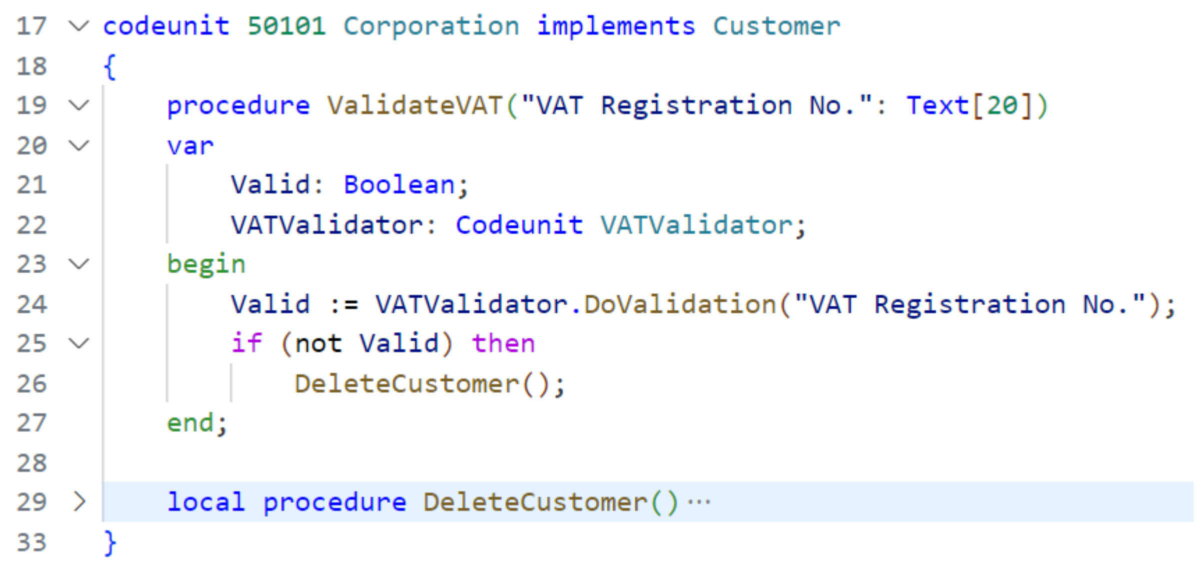
Liskov substitution principle

What’s the idea?

"Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.“ or implementations of interfaces behave similar and can easily be substituted



```
interface Customer
{
    procedure ValidateVAT("VAT Registration No.": Text[20]);
}

codeunit 50100 Company implements Customer
{
    procedure ValidateVAT("VAT Registration No.": Text[20])
    var
        Valid: Boolean;
        VATValidator: Codeunit VATValidator;
    begin
        Valid := VATValidator.DoValidation("VAT Registration No.");
    end;
}
```

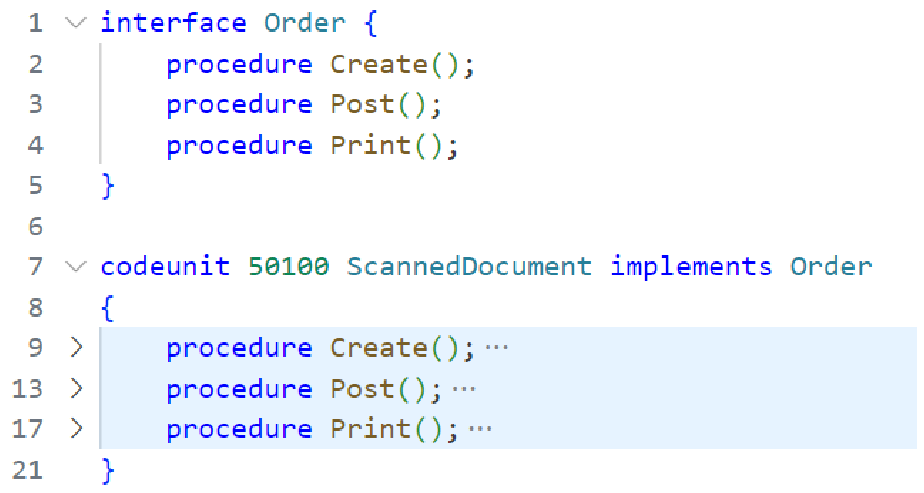


```
codeunit 50101 Corporation implements Customer
{
    procedure ValidateVAT("VAT Registration No.": Text[20])
    var
        Valid: Boolean;
        VATValidator: Codeunit VATValidator;
    begin
        Valid := VATValidator.DoValidation("VAT Registration No.");
        if (not Valid) then
            DeleteCustomer();
    end;

    local procedure DeleteCustomer()···
}
```

Interface segregation principle

What's the idea?

"Clients should not be forced to depend upon interfaces that they do not use.“ or keep your interfaces to the necessary minimum



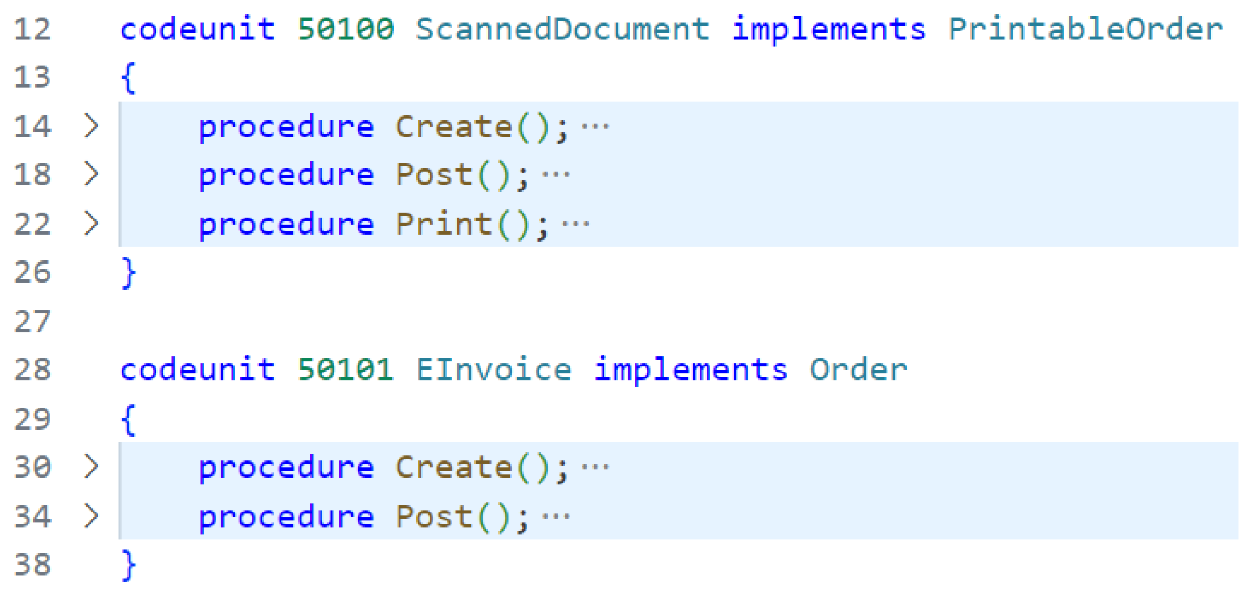
```
interface Order {
    procedure Create();
    procedure Post();
    procedure Print();
}

codeunit 50100 ScannedDocument implements Order
{
    procedure Create();…
    procedure Post();…
    procedure Print();…
}
```

directions lyon emea 2023

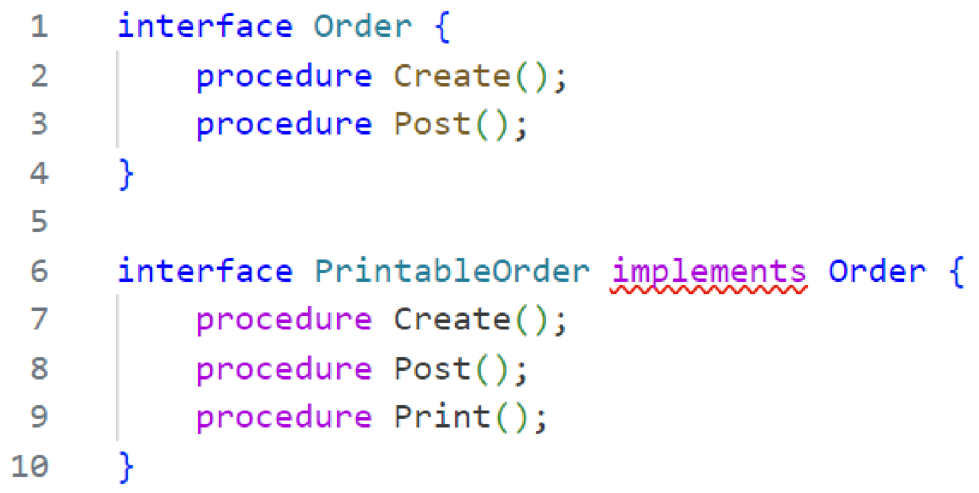
Interface segregation principle

Some wishful thinking

"Clients should not be forced to depend upon interfaces that they do not use." or keep your interfaces to the necessary minimum



```
codeunit 50100 ScannedDocument implements PrintableOrder
{
    procedure Create(); ···
    procedure Post(); ···
    procedure Print(); ···
}

codeunit 50101 EInvoice implements Order
{
    procedure Create(); ···
    procedure Post(); ···
}
```



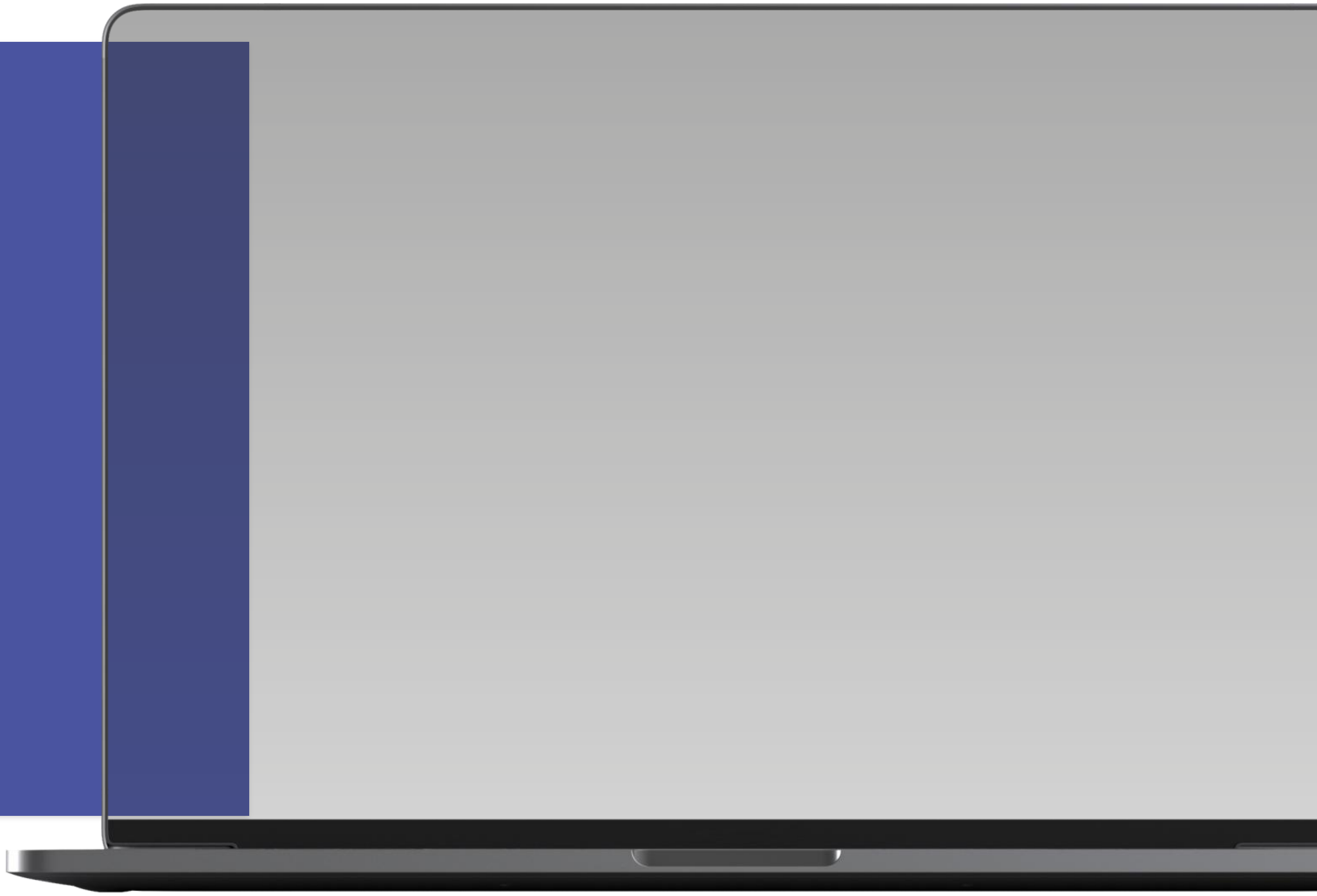
```
interface Order {
    procedure Create();
    procedure Post();
}

interface PrintableOrder implements Order {
    procedure Create();
    procedure Post();
    procedure Print();
}
```

Dependency inversion principle

What’s the idea?

“Depend on abstractions, not concretions”

SOLID in real life
Dependency inversion



Dependency inversion principle

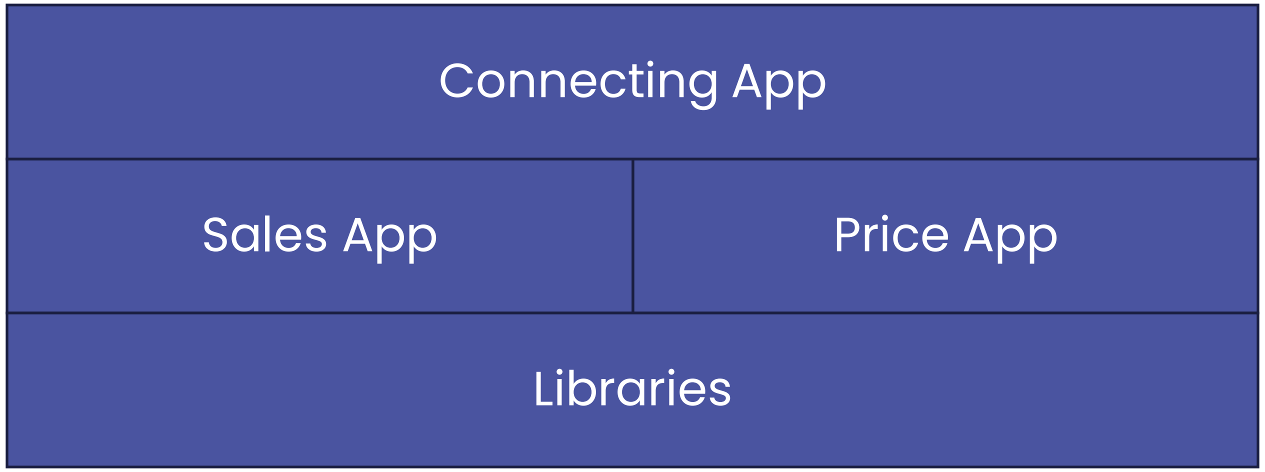
Real life example

- Reduce the Business Central Dependency Graph to a minimum
- Move Connection between apps from Top to Bottom of the graph
- Each App is only Dependent on the Interface Layer
- Decouple Calling Code and Implementation Code

Dependency inversion principle

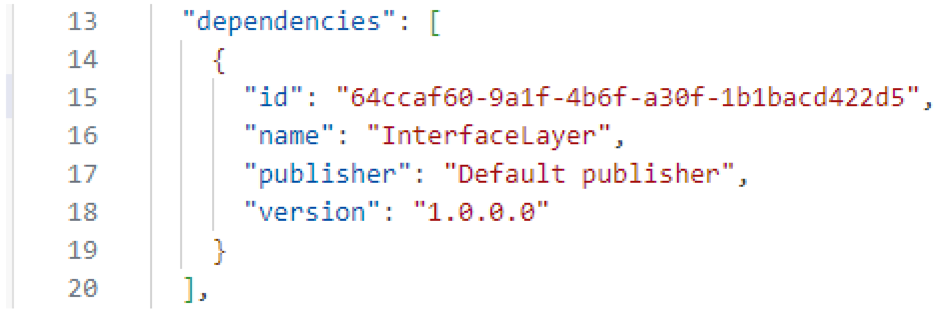
Real life example

```json
"dependencies": [
  {
    "id": "5e398f07-9794-4f7a-9062-18c18fd88924",
    "name": "SalesApp",
    "publisher": "Default publisher",
    "version": "1.0.0.0"
  },
  {
    "id": "e3caba63-19ba-40fc-b3ea-e1ba21b194ff",
    "name": "Libraries",
    "publisher": "Default publisher",
    "version": "1.0.0.0"
  }
],
```

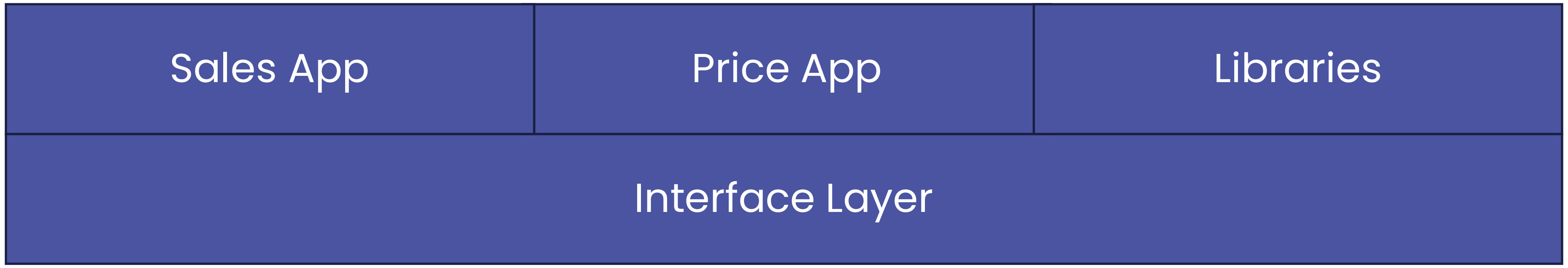


Dependency inversion principle

Real life example



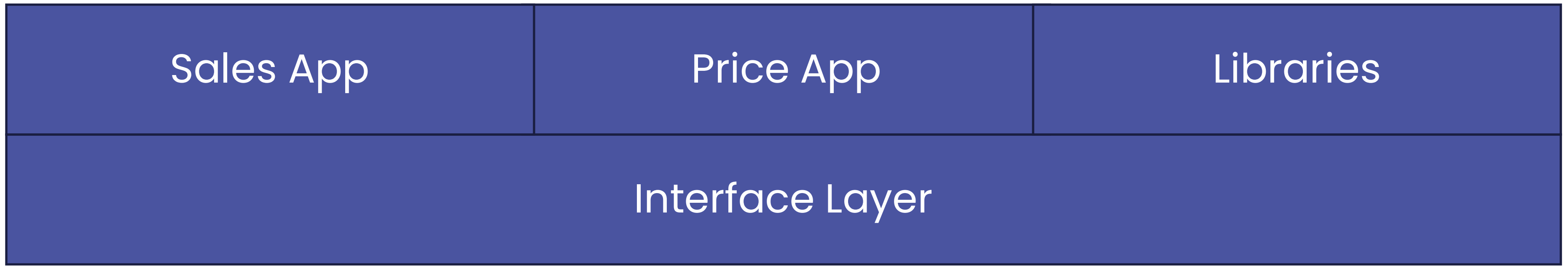
```
"dependencies": [
  {
    "id": "64ccaf60-9a1f-4b6f-a30f-1b1bacd422d5",
    "name": "InterfaceLayer",
    "publisher": "Default publisher",
    "version": "1.0.0.0"
  }
],
```



Dependency inversion principle

Real life example

- Barely no code in Interface Layer
- Easy to update each app
- Interface Layer could be one or multiple apps



Dependency inversion principle

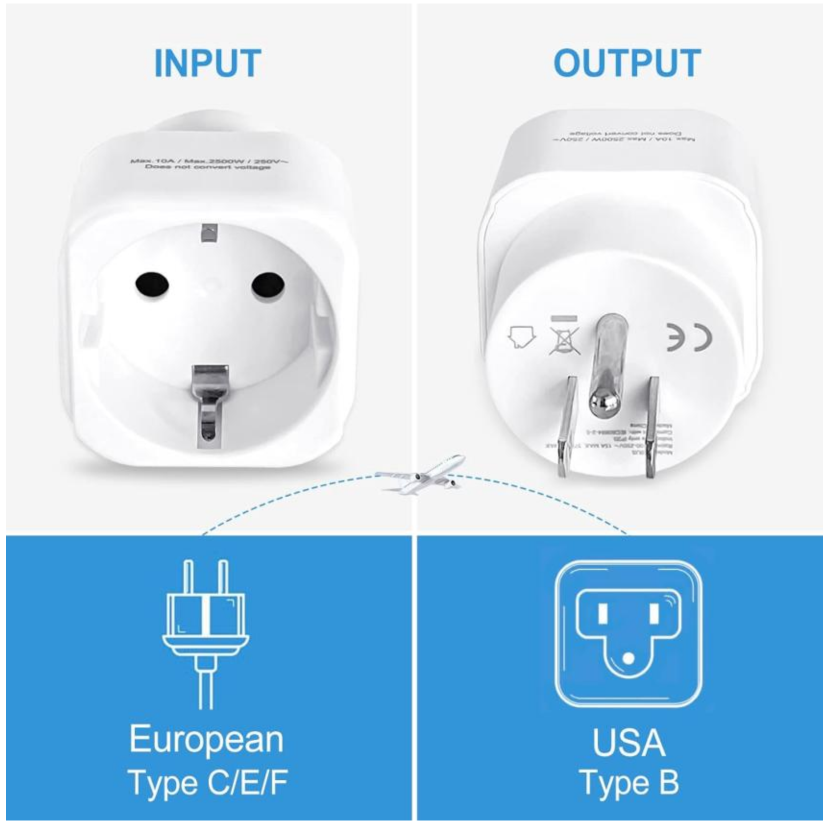
Real life example

LIVE DEMO

directions lyon emea 2023

Adapter

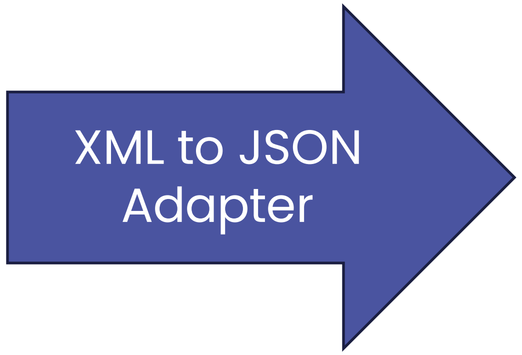
As an example of an object-oriented pattern



Adapter

As an example of an object-oriented pattern

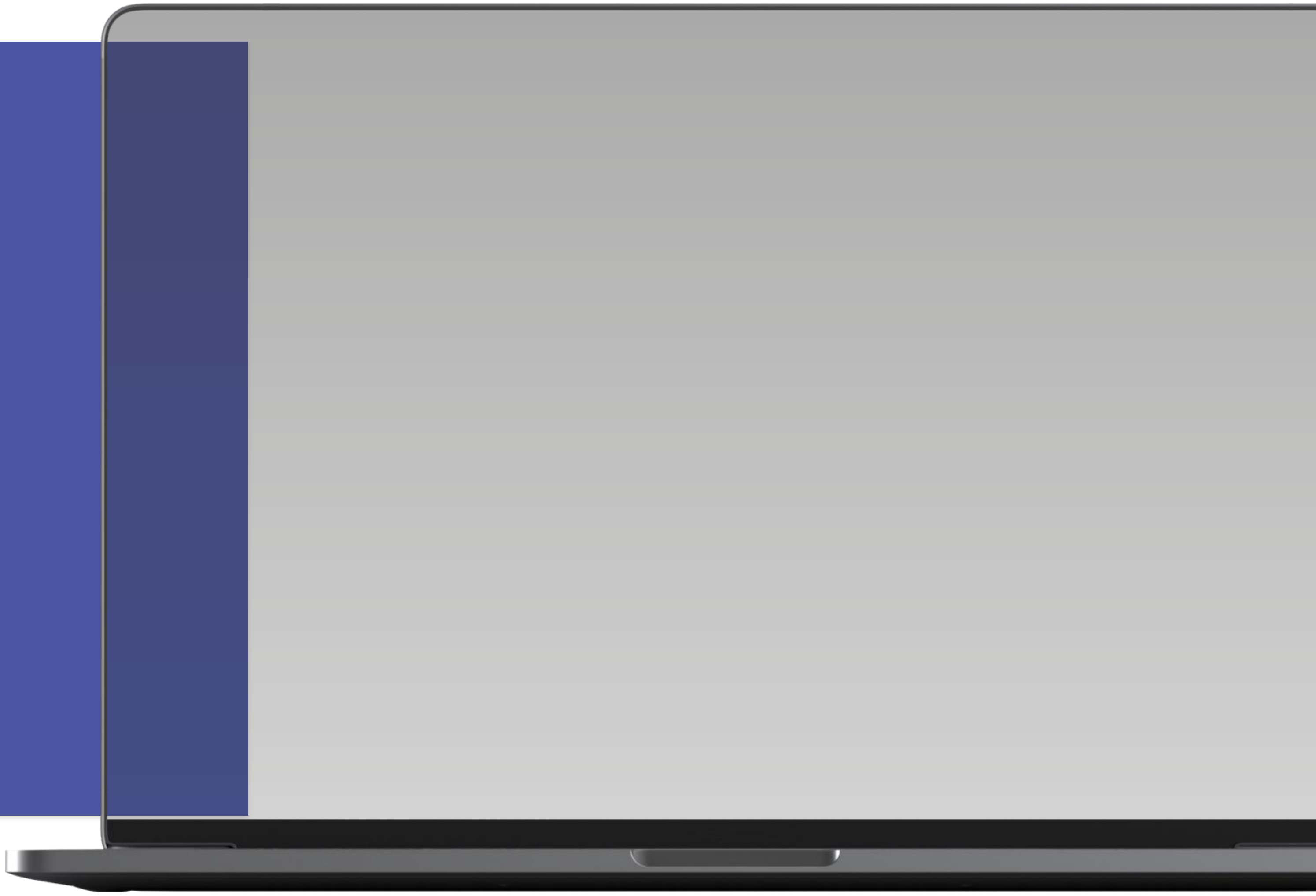
```
<customers>
    <customer no=“1”
        name=“Meier” />
</customers>
```



```
{
    “customers”: [
        {
            “no”: 1,
            “name”: “Meier”
        }
    ]
}
```

directions lyon emea 2023

Adapter in real life ?



Adapter

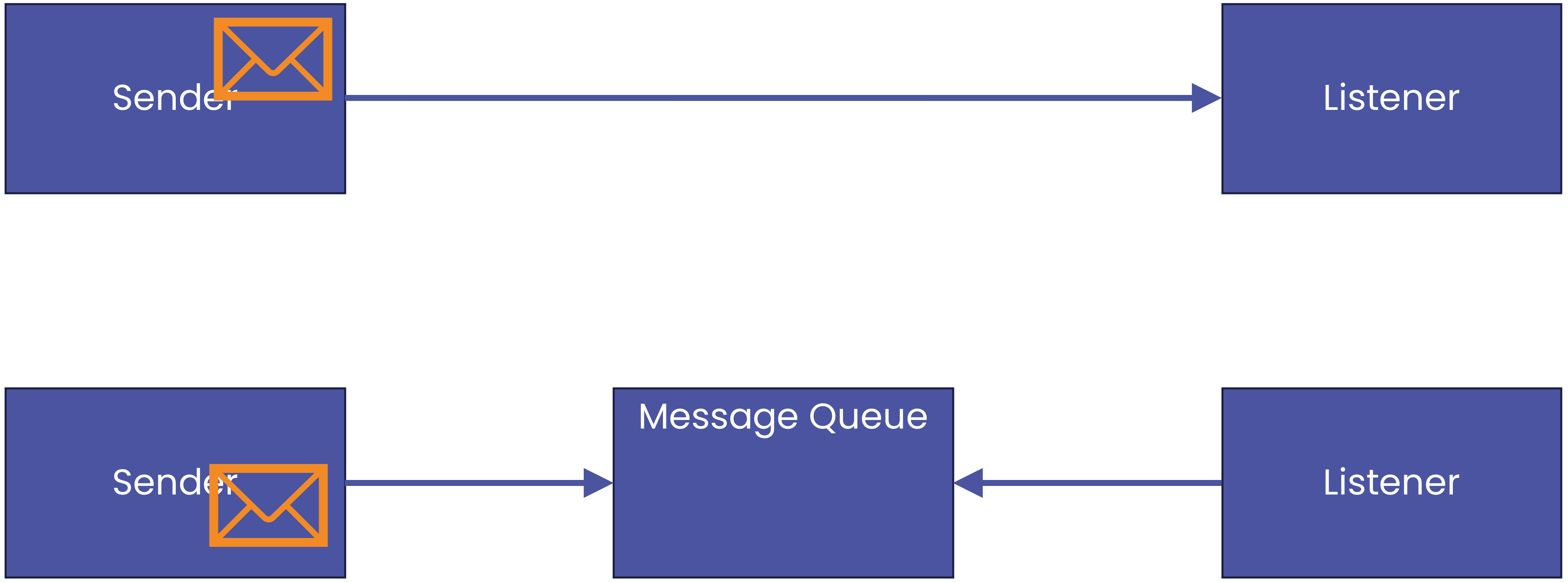
Real life example

directions lyon emea 2023

LIVE DEMO

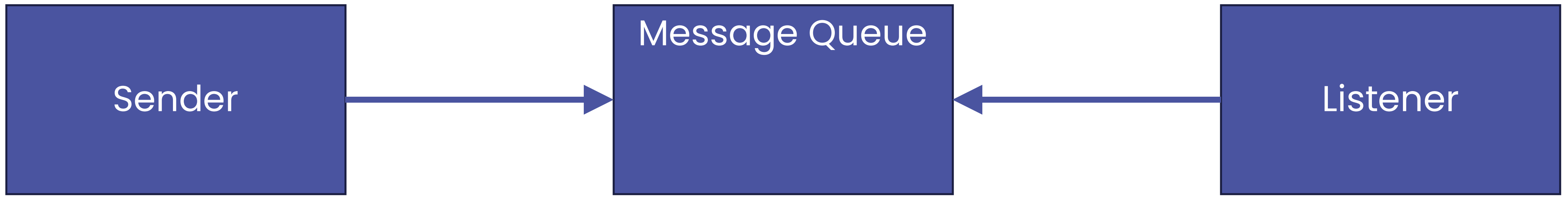
Integrating external components

Architecture pattern “Service Bus”



Integrating external components

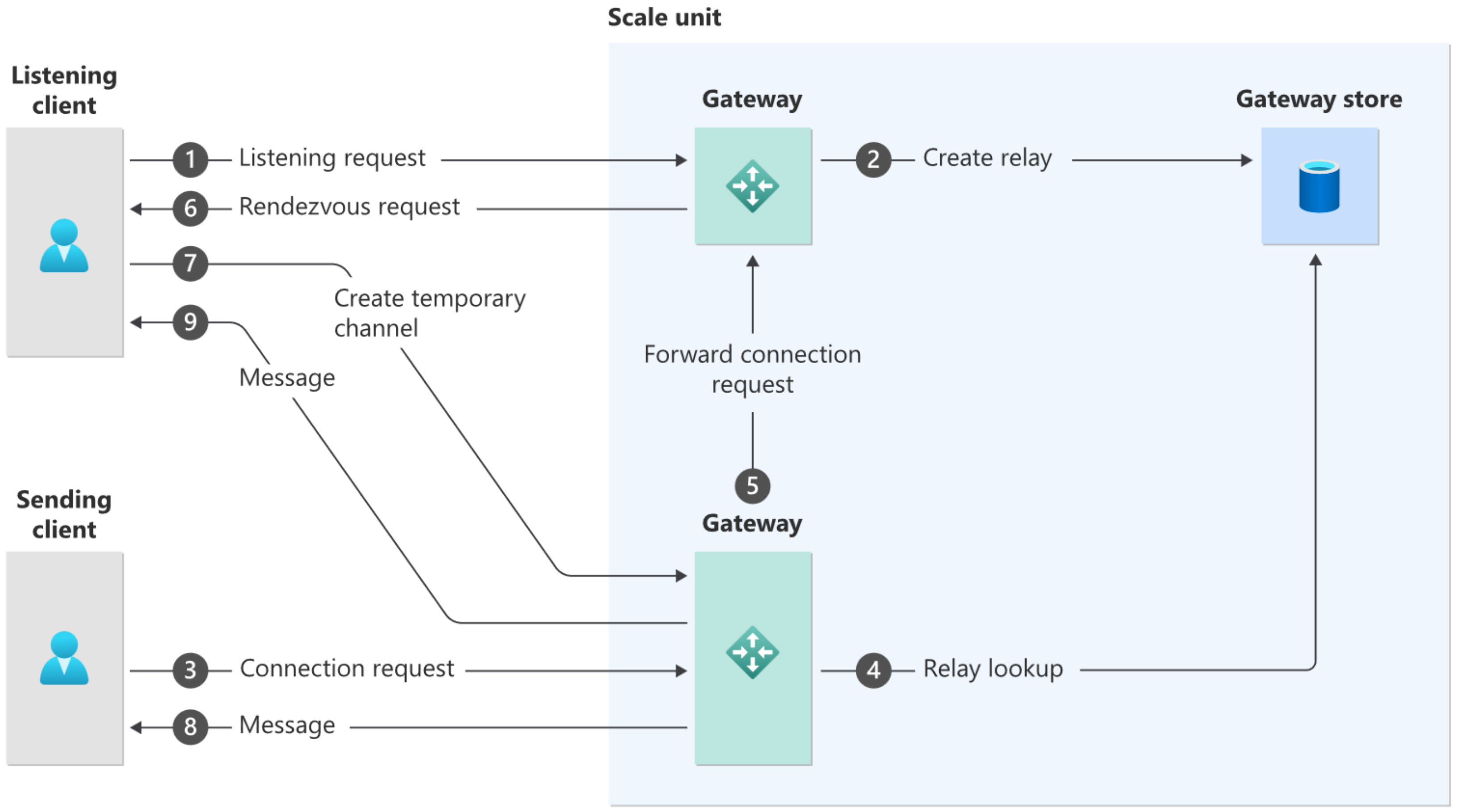
Architecture pattern “Service Bus”



- Easy to support unstable listeners and still have reliable messaging
- Message order can be ensured easily
- Easy to scale: Even with a lot of listeners, the sender can be the same and the message queue can be scaled relatively easily
- Sender and listener are decoupled

Integrating external components

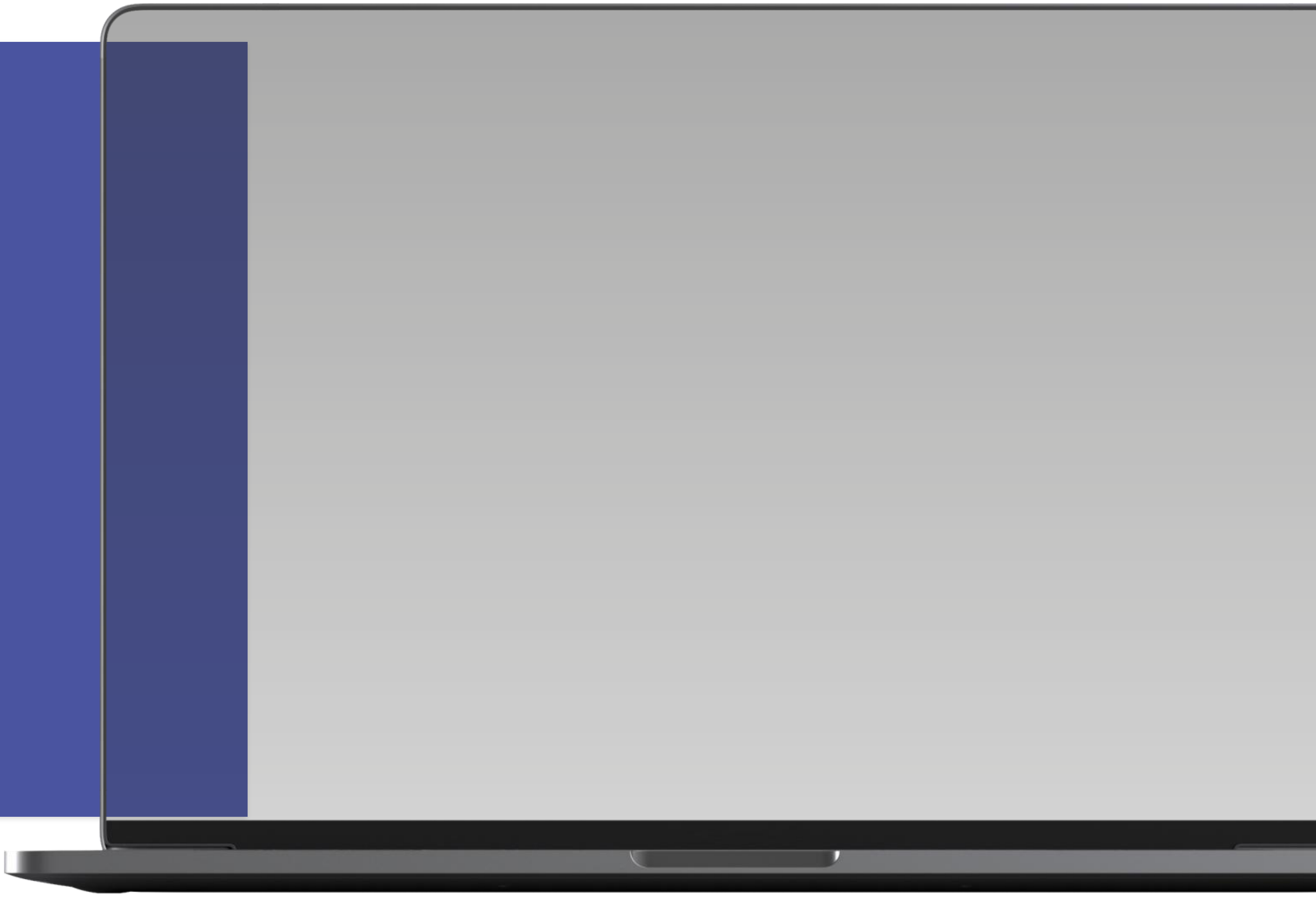
Implementation Azure Relay



Source: https://learn.microsoft.com/en-us/azure/azure-relay/relay-what-is-it

directions lyon
emea 2023

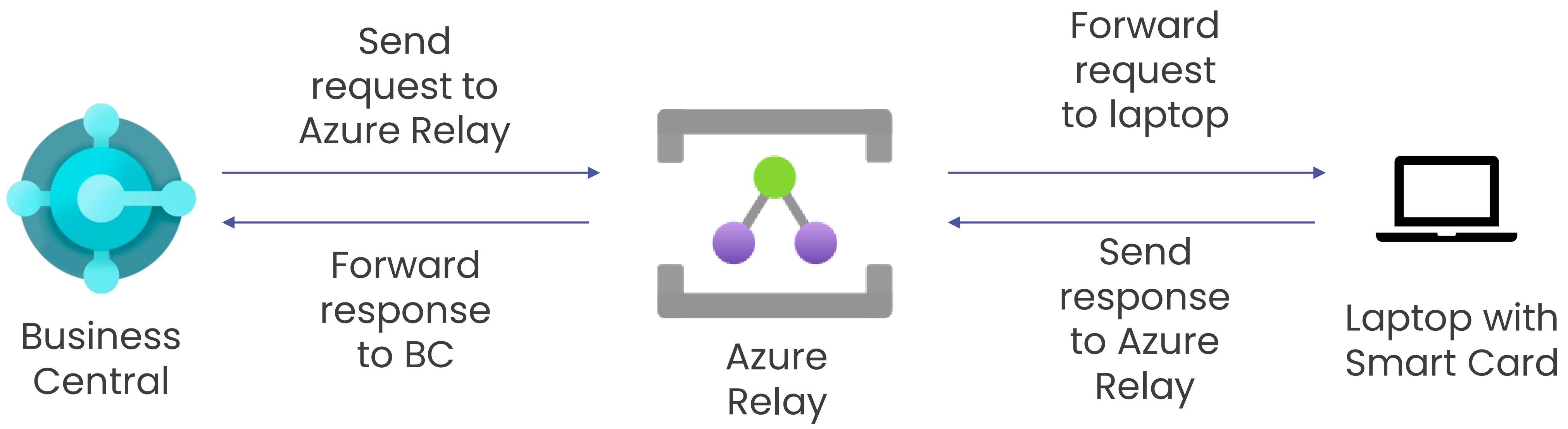
Integrating external components
Azure Relay



directions lyon emea 2023

Access local Hardware from SaaS

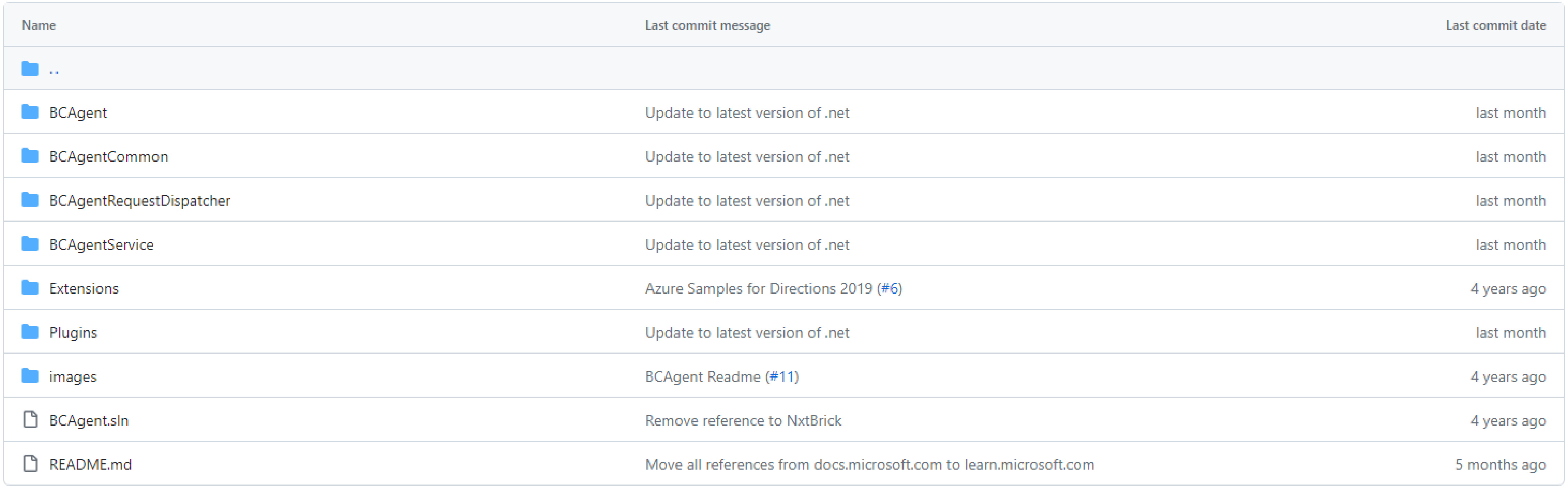
Real life example


Send request to Azure Relay
Forward request to laptop
Business Central
Forward response to BC
Azure Relay
Send response to Azure Relay
Laptop with Smart Card

Access local Hardware from SaaS

Real life example

- Based on https://aka.ms/BCTech BCAgent



Name	Last commit message	Last commit date
..		
BCAgent	Update to latest version of .net	last month
BCAgentCommon	Update to latest version of .net	last month
BCAgentRequestDispatcher	Update to latest version of .net	last month
BCAgentService	Update to latest version of .net	last month
Extensions	Azure Samples for Directions 2019 (#6)	4 years ago
Plugins	Update to latest version of .net	last month
images	BCAgent Readme (#11)	4 years ago
BCAgent.sln	Remove reference to NxtBrick	4 years ago
README.md	Move all references from docs.microsoft.com to learn.microsoft.com	5 months ago

Access local Hardware from SaaS

Real life example

LIVE DEMO

directions lyon emea 2023

Give us Feedback!

and help us improve!

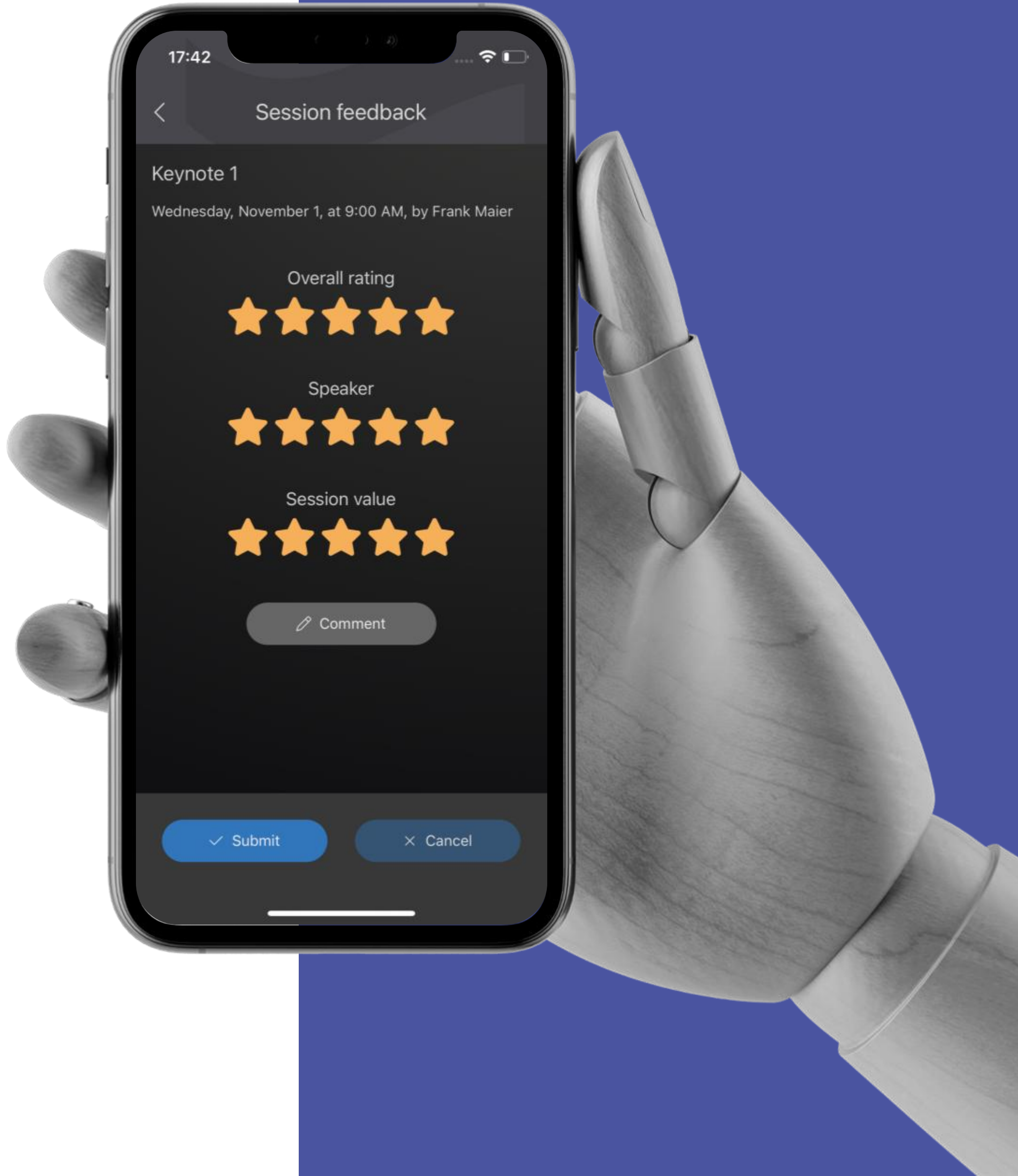
Please rate our session in the Conference App!

and leave a constructive comment.

Win a 100€ gift card

one ticket for every session feedback!

Thank you !



directions lyon
emea 2023

Which question can we answer?