devops HEROES

encase LABS

La [sottile] arte di testare le Durable Functions!!

Massimo Bonanni

Technical Trainer @ Microsoft

massimo.bonanni@microsoft.com

Codice Sessione 104

Sponsor & Org

UNIVERSITÀ DEGLI STUDI DI PARMA

engage LABS

innprojekt

iSolutions

daniela malvisi COMMUNICATION

DRILLING PERGEMINE CONTRACTOR

LIME ware

getlatestversion.it

Azure Functions testing fundamentals!!!

The issue

Testing the components of an Enterprise solution is essential for ensuring **reliability** and **maintainability**, as it allows for **early identification of issues**, easier debugging, and seamless integration into complex, event-driven architectures.

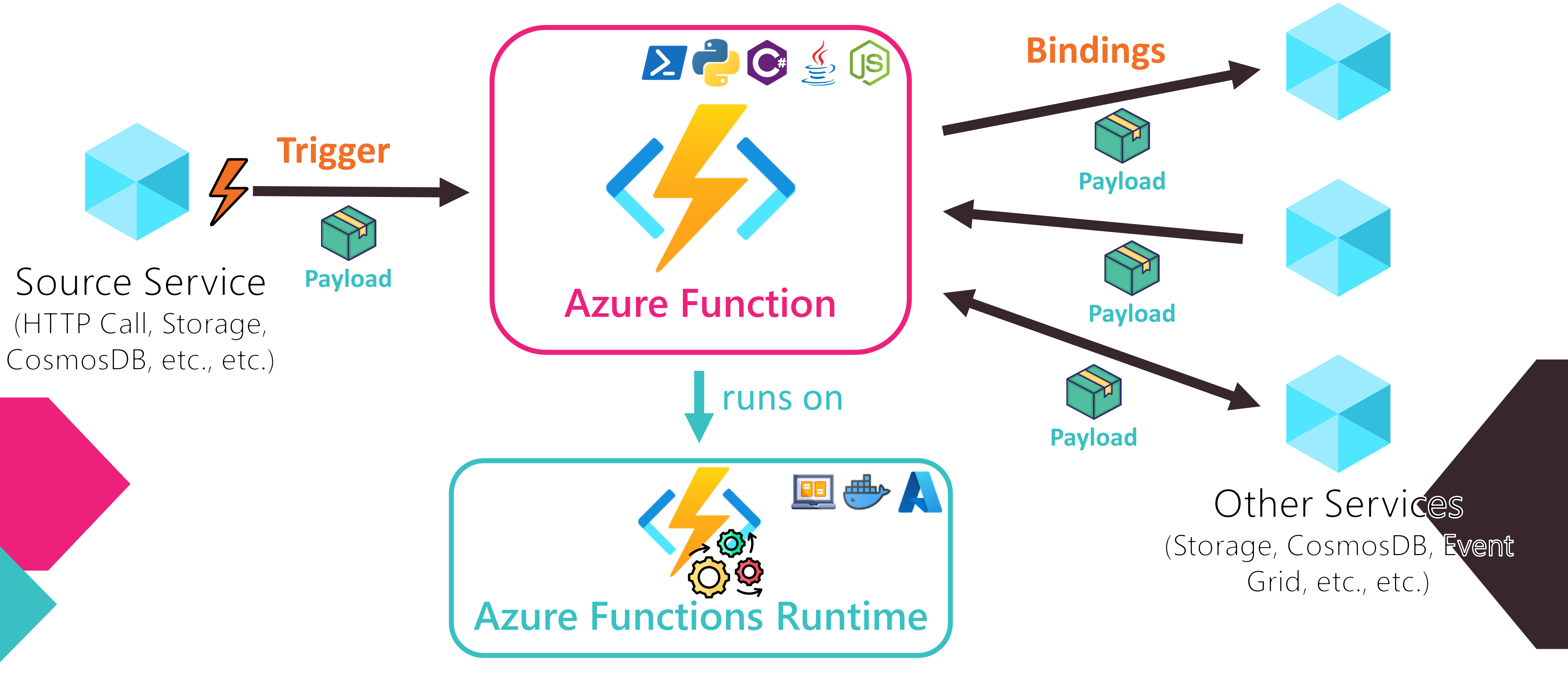
What is a unit test

In a **unit test** you **invoke** a piece of your code with a set of parameters, and you checks the correctness of its behavior.

In a **unit test** you **must** substitute all your external reference with a **mock** or **stub**.

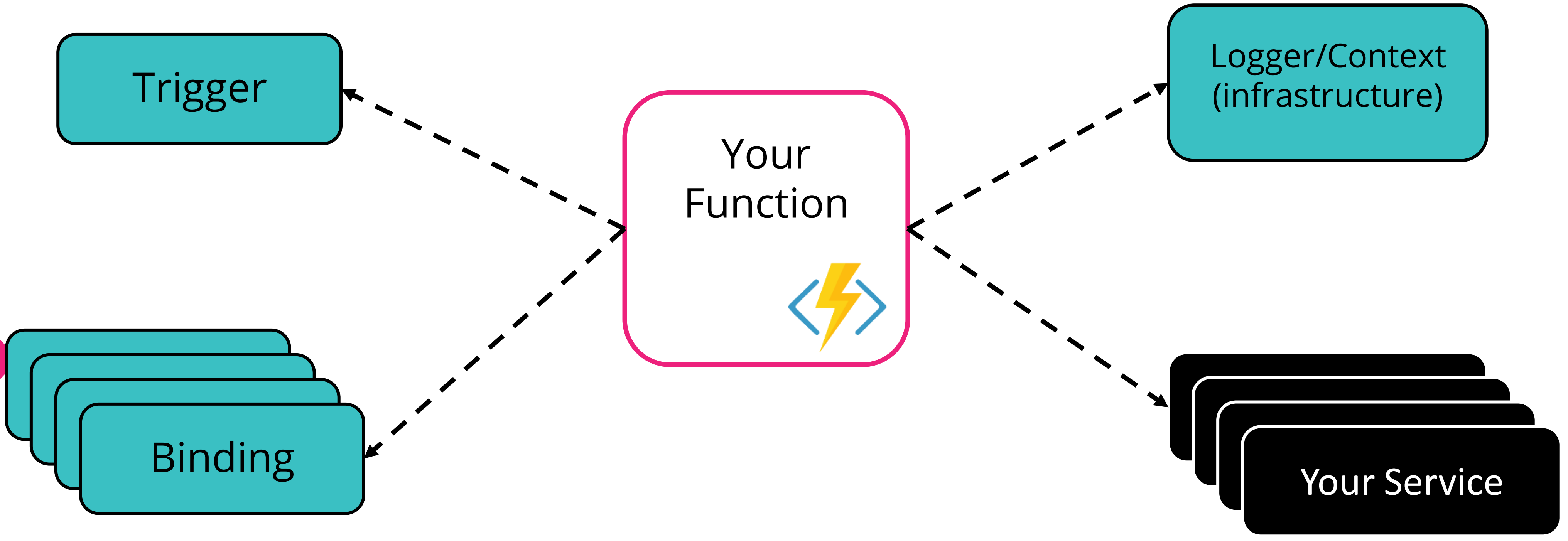
Mock is for the software what a ***dummy*** *is for a car crash test (you don't test a car with a human being inside…I hope!!)*

Azure Functions components

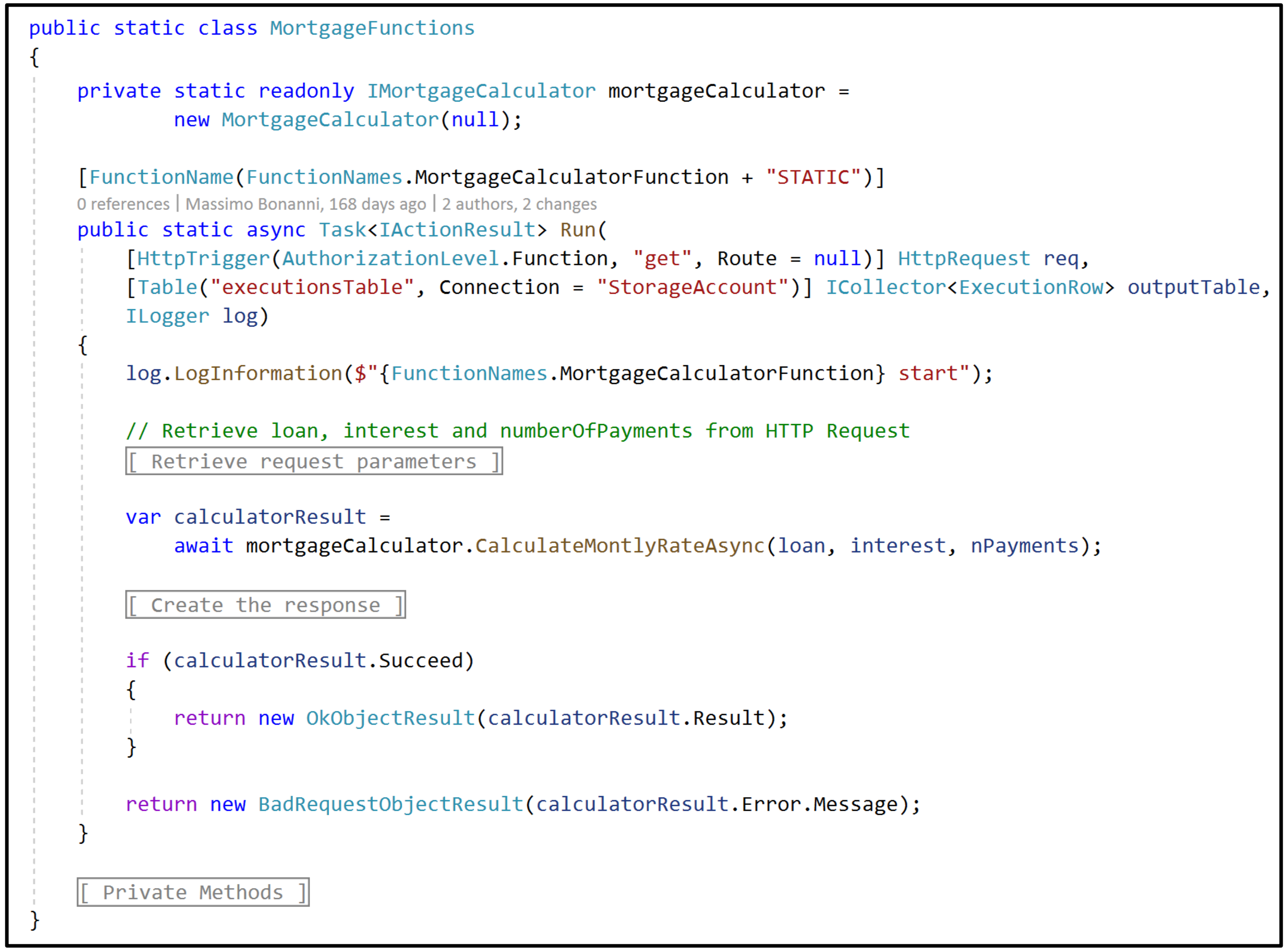


Azure Functions Dependencies

You **should implement** your Azure Functions to allow you to use mock/stub for all external reference!

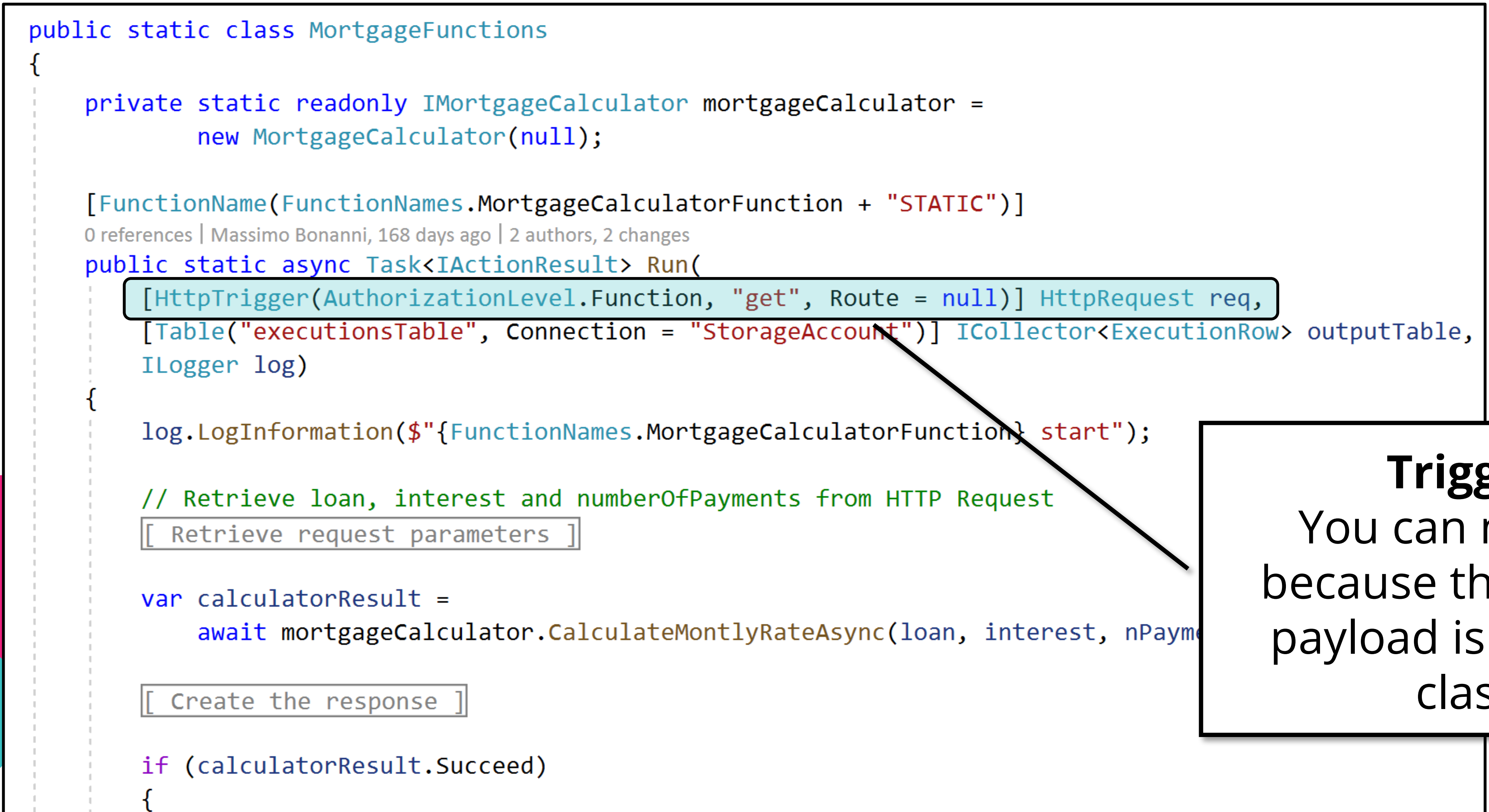


Azure Function … untestable!!



```
public static class MortgageFunctions
{
    private static readonly IMortgageCalculator mortgageCalculator =
            new MortgageCalculator(null);

    [FunctionName(FunctionNames.MortgageCalculatorFunction + "STATIC")]
    0 references | Massimo Bonanni, 168 days ago | 2 authors, 2 changes
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", Route = null)] HttpRequest req,
        [Table("executionsTable", Connection = "StorageAccount")] ICollector<ExecutionRow> outputTable,
        ILogger log)
    {
        log.LogInformation($"{FunctionNames.MortgageCalculatorFunction} start");

        // Retrieve loan, interest and numberOfPayments from HTTP Request
        [ Retrieve request parameters ]

        var calculatorResult =
            await mortgageCalculator.CalculateMontlyRateAsync(loan, interest, nPayments);

        [ Create the response ]

        if (calculatorResult.Succeed)
        {
            return new OkObjectResult(calculatorResult.Result);
        }

        return new BadRequestObjectResult(calculatorResult.Error.Message);
    }

    [ Private Methods ]
}
```

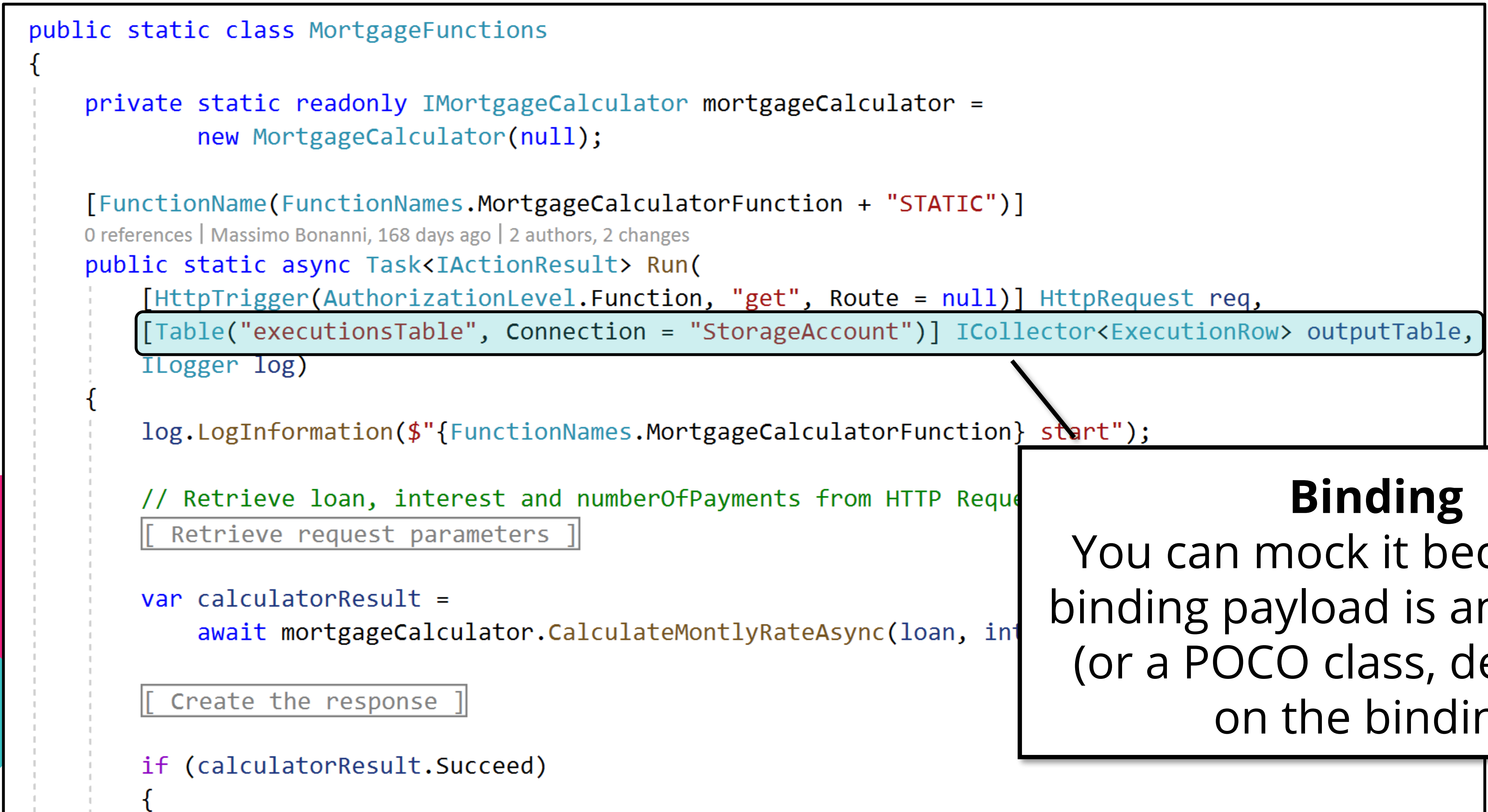
Azure Function … trigger!!



```
public static class MortgageFunctions
{
    private static readonly IMortgageCalculator mortgageCalculator =
            new MortgageCalculator(null);

    [FunctionName(FunctionNames.MortgageCalculatorFunction + "STATIC")]
    0 references | Massimo Bonanni, 168 days ago | 2 authors, 2 changes
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", Route = null)] HttpRequest req,
        [Table("executionsTable", Connection = "StorageAccount")] ICollector<ExecutionRow> outputTable,
        ILogger log)
    {
        log.LogInformation($"{FunctionNames.MortgageCalculatorFunction} start");

        // Retrieve loan, interest and numberOfPayments from HTTP Request
        [ Retrieve request parameters ]

        var calculatorResult =
            await mortgageCalculator.CalculateMontlyRateAsync(loan, interest, nPayme

        [ Create the response ]

        if (calculatorResult.Succeed)
        {
```

Trigger
You can mock it because the trigger payload is a POCO class

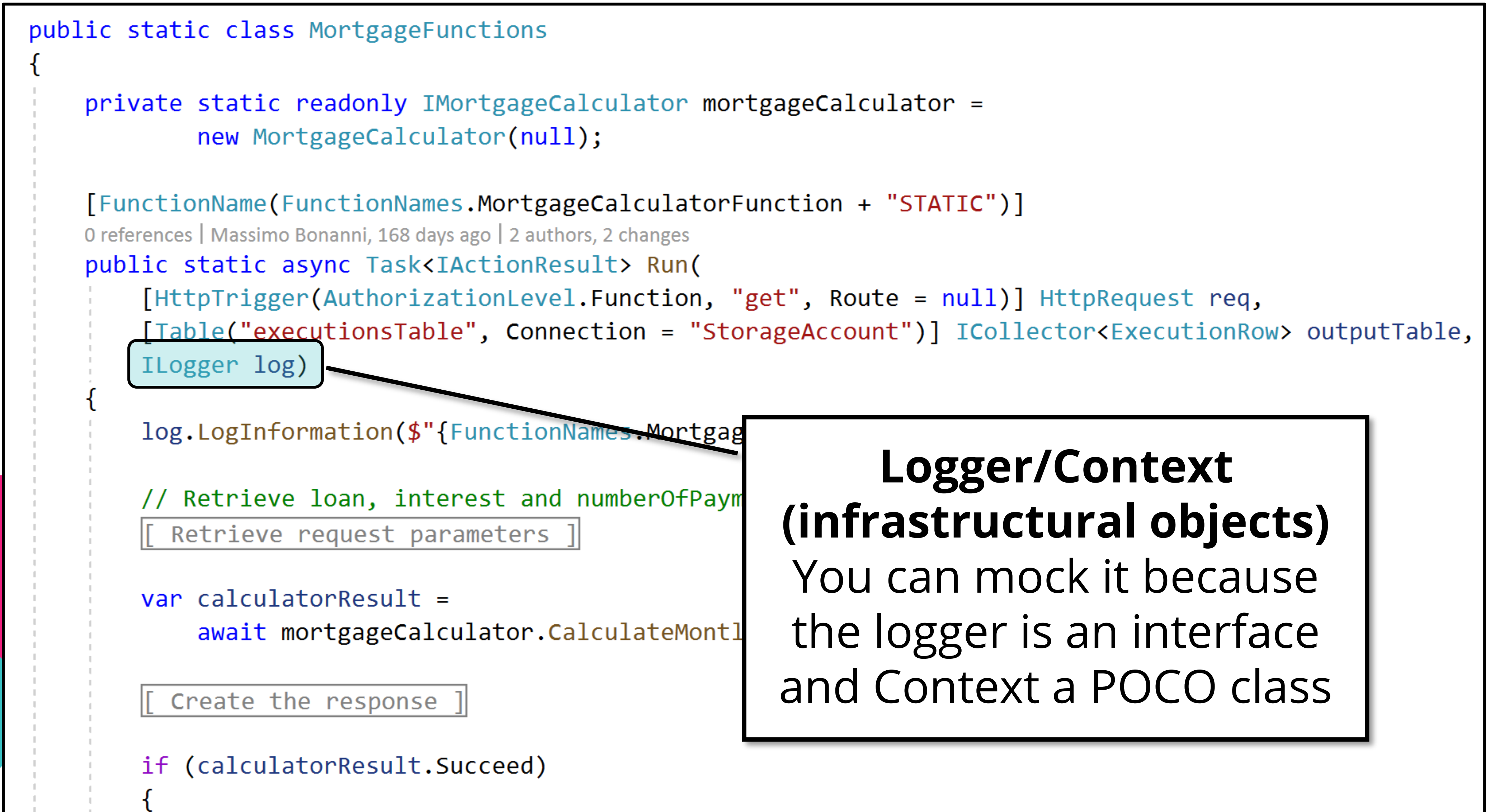
Azure Function … bindings!!



```
public static class MortgageFunctions
{
    private static readonly IMortgageCalculator mortgageCalculator =
            new MortgageCalculator(null);

    [FunctionName(FunctionNames.MortgageCalculatorFunction + "STATIC")]
    0 references | Massimo Bonanni, 168 days ago | 2 authors, 2 changes
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", Route = null)] HttpRequest req,
        [Table("executionsTable", Connection = "StorageAccount")] ICollector<ExecutionRow> outputTable,
        ILogger log)
    {
        log.LogInformation($"{FunctionNames.MortgageCalculatorFunction} start");

        // Retrieve loan, interest and numberOfPayments from HTTP Reque
        [ Retrieve request parameters ]

        var calculatorResult =
            await mortgageCalculator.CalculateMontlyRateAsync(loan, int

        [ Create the response ]

        if (calculatorResult.Succeed)
        {
```

Binding
You can mock it because the binding payload is an interface (or a POCO class, depending on the binding)

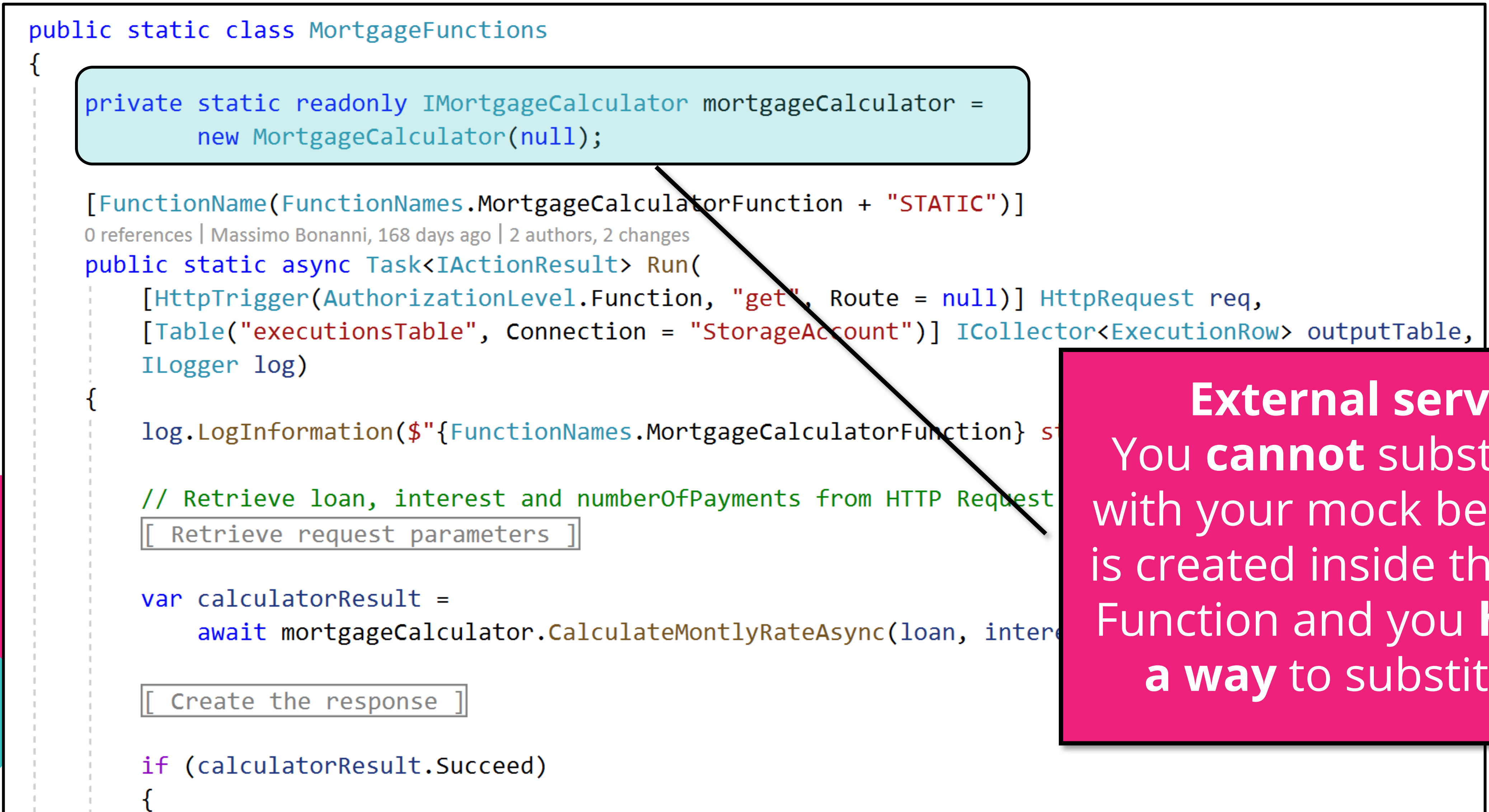
Azure Function … logger!!



```
public static class MortgageFunctions
{
    private static readonly IMortgageCalculator mortgageCalculator =
        new MortgageCalculator(null);

    [FunctionName(FunctionNames.MortgageCalculatorFunction + "STATIC")]
    0 references | Massimo Bonanni, 168 days ago | 2 authors, 2 changes
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", Route = null)] HttpRequest req,
        [Table("executionsTable", Connection = "StorageAccount")] ICollector<ExecutionRow> outputTable,
        ILogger log)
    {
        log.LogInformation($"{FunctionNames.Mortgag

        // Retrieve loan, interest and numberOfPaym
        [ Retrieve request parameters ]

        var calculatorResult =
            await mortgageCalculator.CalculateMontl

        [ Create the response ]

        if (calculatorResult.Succeed)
        {
```

Logger/Context (infrastructural objects)
You can mock it because the logger is an interface and Context a POCO class

Azure Function … your service!!



```
public static class MortgageFunctions
{
    private static readonly IMortgageCalculator mortgageCalculator =
            new MortgageCalculator(null);

    [FunctionName(FunctionNames.MortgageCalculatorFunction + "STATIC")]
    0 references | Massimo Bonanni, 168 days ago | 2 authors, 2 changes
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", Route = null)] HttpRequest req,
        [Table("executionsTable", Connection = "StorageAccount")] ICollector<ExecutionRow> outputTable,
        ILogger log)
    {
        log.LogInformation($"{FunctionNames.MortgageCalculatorFunction} st

        // Retrieve loan, interest and numberOfPayments from HTTP Request
        [ Retrieve request parameters ]

        var calculatorResult =
            await mortgageCalculator.CalculateMontlyRateAsync(loan, intere

        [ Create the response ]

        if (calculatorResult.Succeed)
        {
```

External service
You **cannot** substitute it with your mock because it is created inside the Azure Function and you **haven't a way** to substitute it

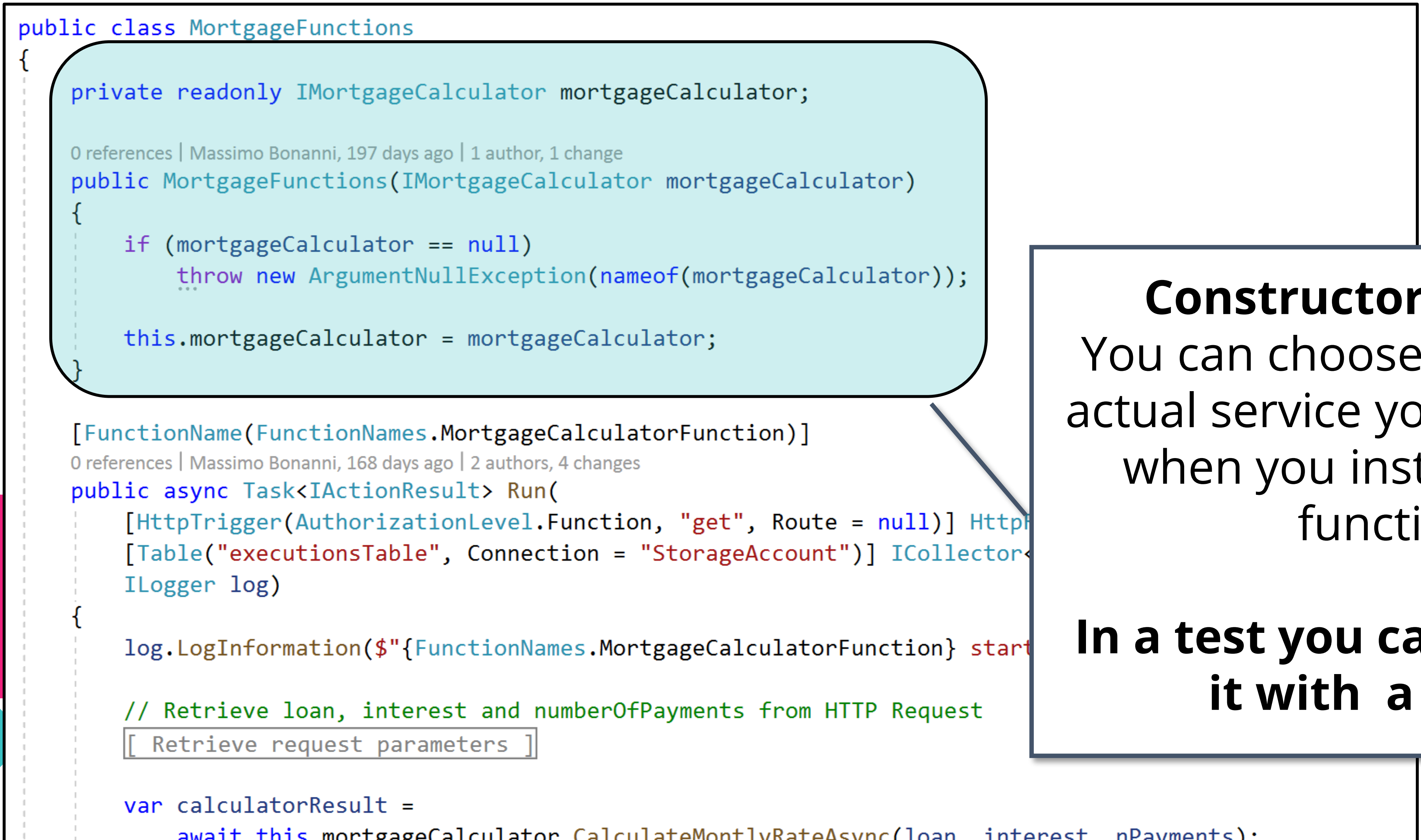
Make your Azure Function testable!!!

The solution of your problem is: **Dependency Injection**!!

Azure Functions Runtime is based on .NET and support the same ASP.NET Dependency Injection!!!

Using Dependency Injection, you provide a way to substitute your services with a mock!

Azure Function … testable!!

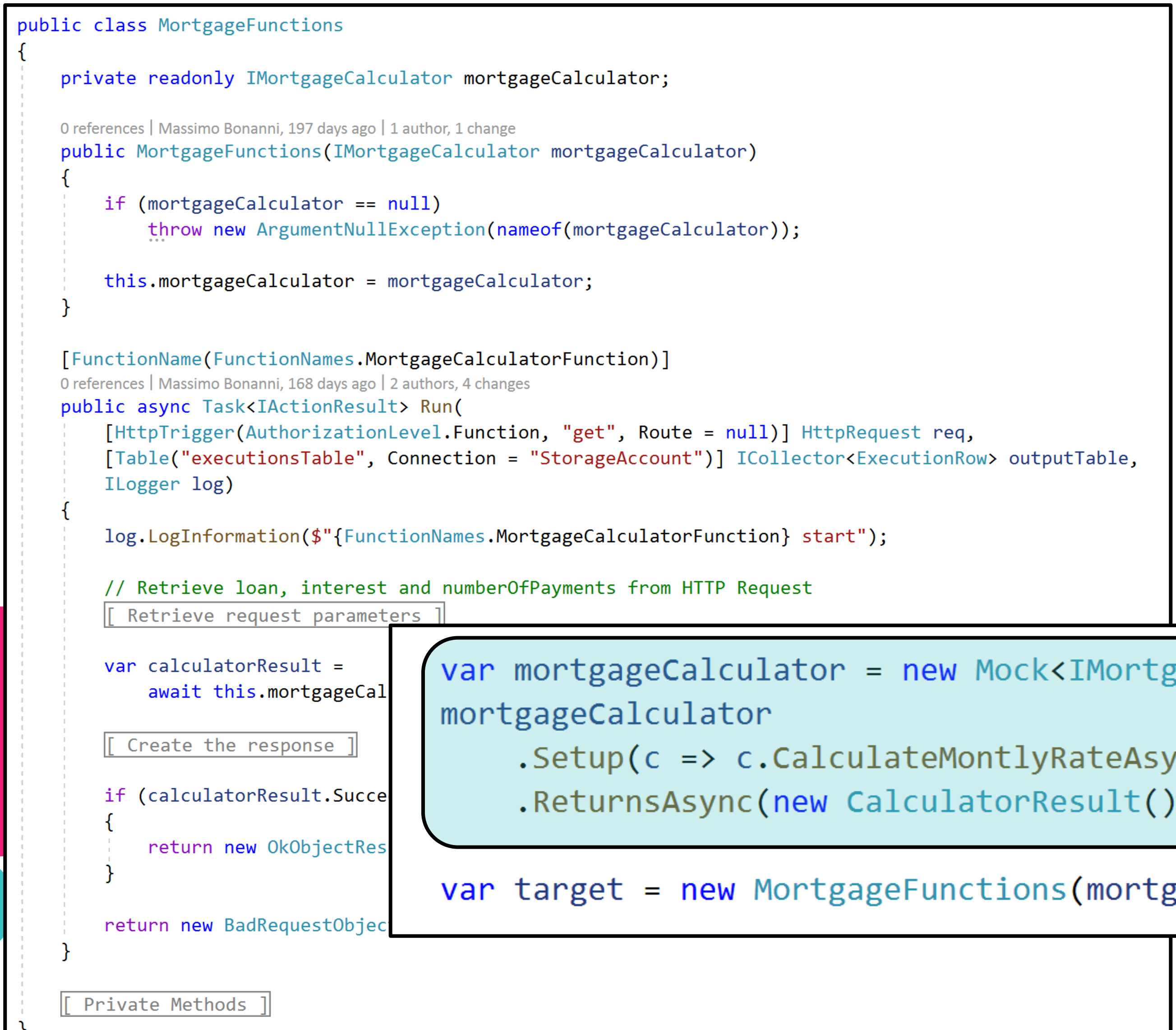


```
public class MortgageFunctions
{
    private readonly IMortgageCalculator mortgageCalculator;

    0 references | Massimo Bonanni, 197 days ago | 1 author, 1 change
    public MortgageFunctions(IMortgageCalculator mortgageCalculator)
    {
        if (mortgageCalculator == null)
            throw new ArgumentNullException(nameof(mortgageCalculator));

        this.mortgageCalculator = mortgageCalculator;
    }

    [FunctionName(FunctionNames.MortgageCalculatorFunction)]
    0 references | Massimo Bonanni, 168 days ago | 2 authors, 4 changes
    public async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", Route = null)] HttpR
        [Table("executionsTable", Connection = "StorageAccount")] ICollector<
        ILogger log)
    {
        log.LogInformation($"{FunctionNames.MortgageCalculatorFunction} start

        // Retrieve loan, interest and numberOfPayments from HTTP Request
        [ Retrieve request parameters ]

        var calculatorResult =
            await this.mortgageCalculator.CalculateMontlyRateAsync(loan, interest, nPayments);
```

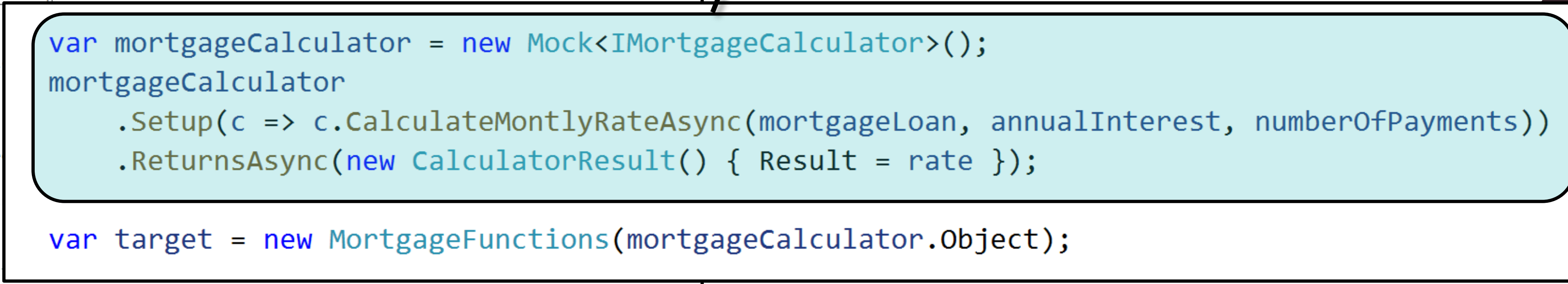
Constructor Injection
You can choose what kind of actual service you want to use when you instantiate the function.

In a test you can substitute it with a mock!!

Azure Function … how to use mock!!



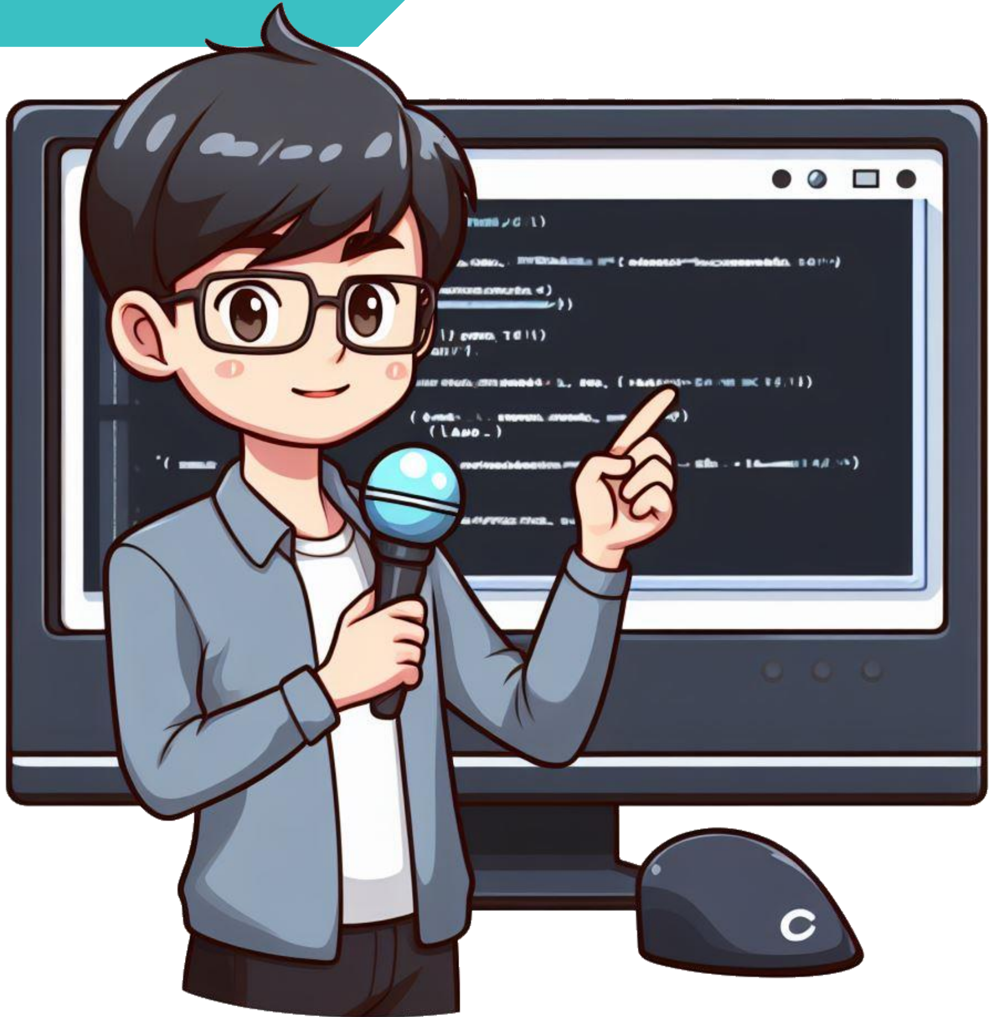
```
public class MortgageFunctions
{
    private readonly IMortgageCalculator mortgageCalculator;

    0 references | Massimo Bonanni, 197 days ago | 1 author, 1 change
    public MortgageFunctions(IMortgageCalculator mortgageCalculator)
    {
        if (mortgageCalculator == null)
            throw new ArgumentNullException(nameof(mortgageCalculator));

        this.mortgageCalculator = mortgageCalculator;
    }

    [FunctionName(FunctionNames.MortgageCalculatorFunction)]
    0 references | Massimo Bonanni, 168 days ago | 2 authors, 4 changes
    public async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", Route = null)] HttpRequest req,
        [Table("executionsTable", Connection = "StorageAccount")] ICollector<ExecutionRow> outputTable,
        ILogger log)
    {
        log.LogInformation($"{FunctionNames.MortgageCalculatorFunction} start");

        // Retrieve loan, interest and numberOfPayments from HTTP Request
        [ Retrieve request parameters ]

        var calculatorResult =
            await this.mortgageCal

        [ Create the response ]

        if (calculatorResult.Succe
        {
            return new OkObjectRes
        }

        return new BadRequestObjec
    }

    [ Private Methods ]
}
```



```
var mortgageCalculator = new Mock<IMortgageCalculator>();
mortgageCalculator
    .Setup(c => c.CalculateMontlyRateAsync(mortgageLoan, annualInterest, numberOfPayments))
    .ReturnsAsync(new CalculatorResult() { Result = rate });

var target = new MortgageFunctions(mortgageCalculator.Object);
```

Mock

Create a mock to use in the test!!



Testing a function!!!

Testing Azure Durable Functions!!

What are Durable Functions?

Durable Functions are Azure Functions!!!

Azure Functions Extension

- Based on Azure Functions
- Adds new Triggers and Bindings
- Manages state, checkpoints, and restarts

Durable Task Framework

- Long running persistent workflows in C#
- Used within various teams at Microsoft to reliably orchestrate long running operations

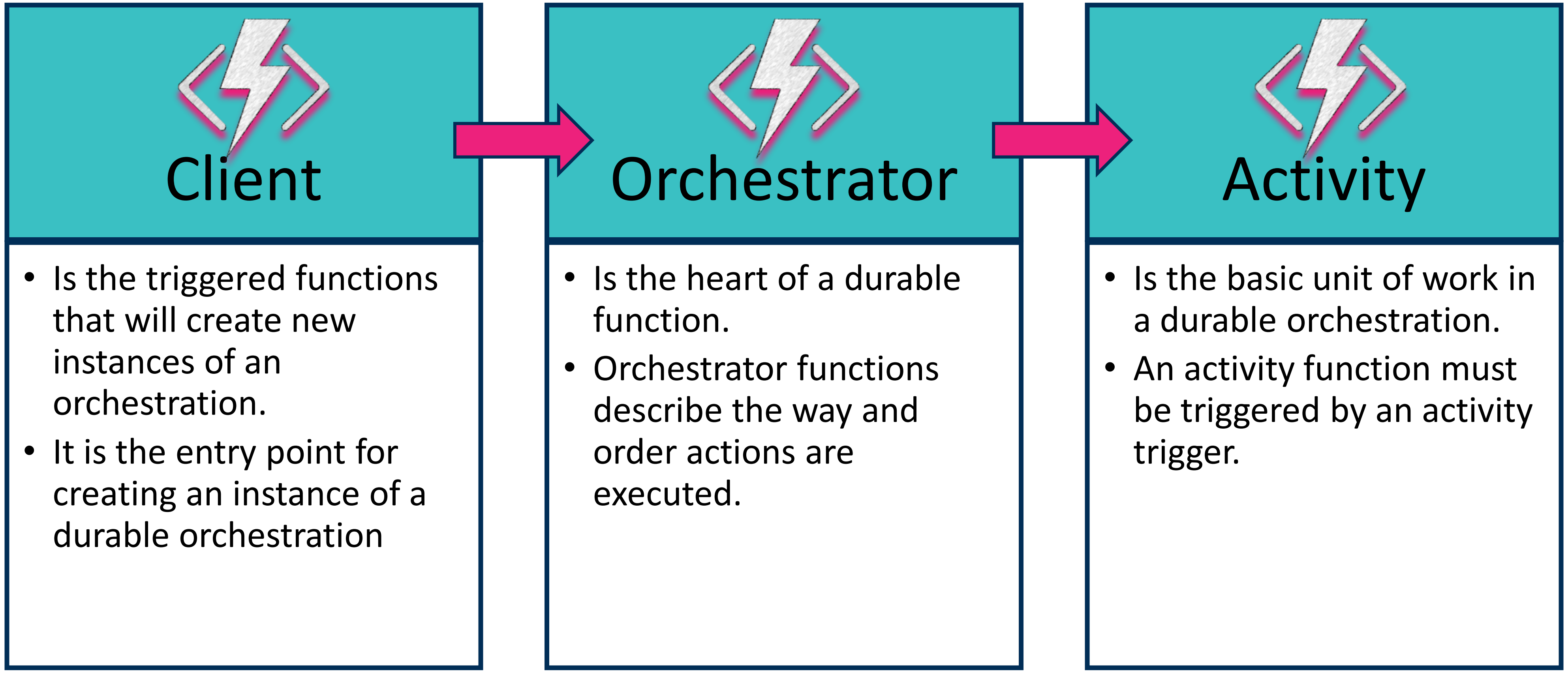
Languages

- C#
- JavaScript
- Java
- Python
- Powershell

Function Types

- Client
- Orchestrator
- Activity
- Entity (no Powershell or Java or .NET isolated)

Types of functions



Client

- Is the triggered functions that will create new instances of an orchestration.
- It is the entry point for creating an instance of a durable orchestration

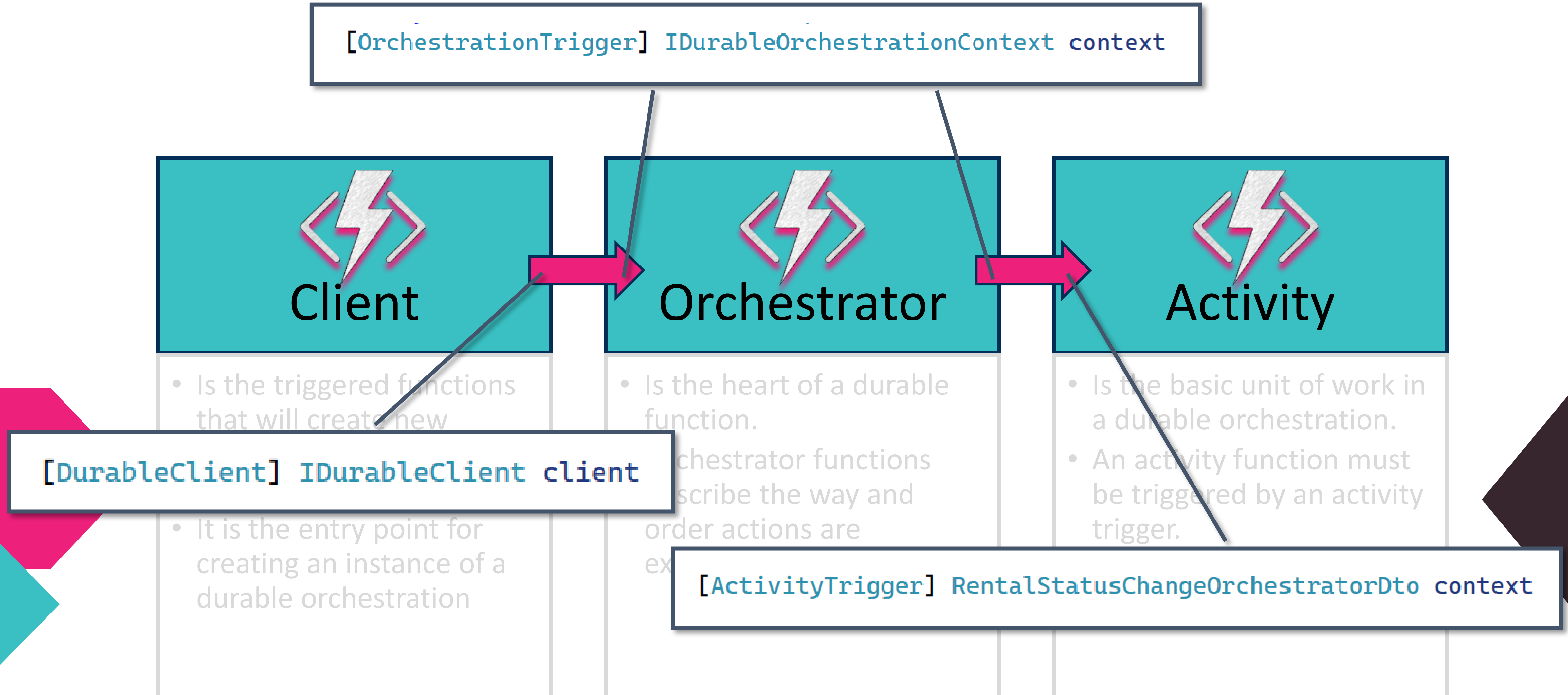
Orchestrator

- Is the heart of a durable function.
- Orchestrator functions describe the way and order actions are executed.

Activity

- Is the basic unit of work in a durable orchestration.
- An activity function must be triggered by an activity trigger.

Types of functions : testability



```
[OrchestrationTrigger] IDurableOrchestrationContext context
```

```
[DurableClient] IDurableClient client
```

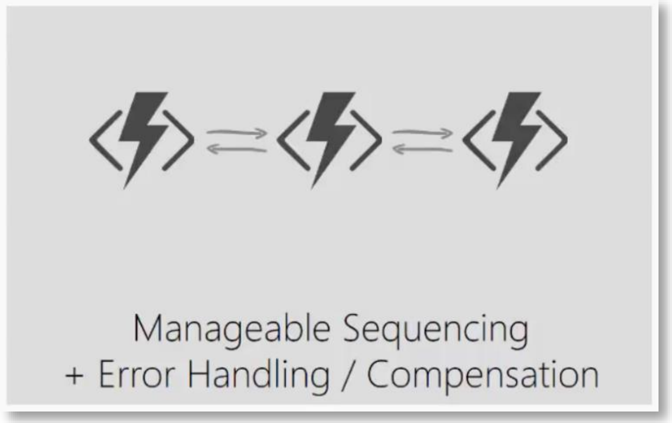
```
[ActivityTrigger] RentalStatusChangeOrchestratorDto context
```

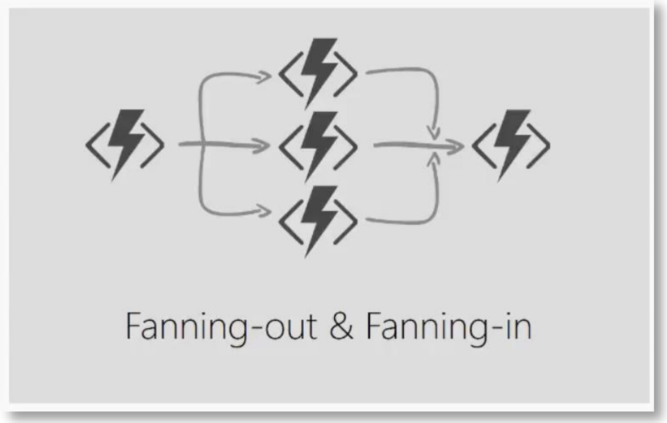
- Is the triggered functions that will create new
- It is the entry point for creating an instance of a durable orchestration

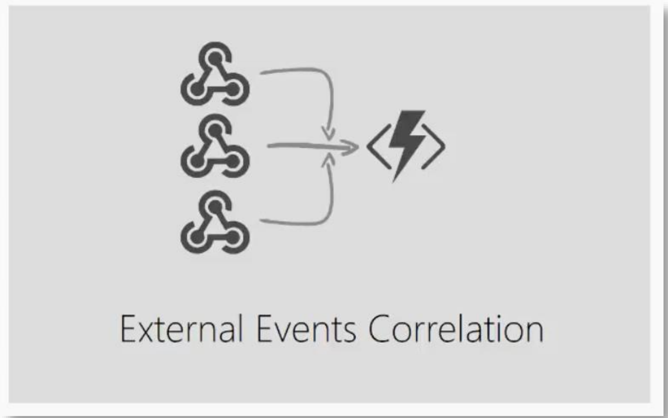
- Is the heart of a durable function.
- chestrator functions scribe the way and order actions are ex

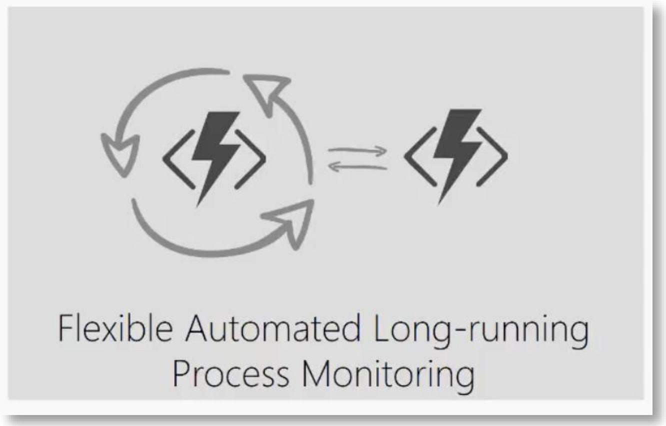
- Is the basic unit of work in a durable orchestration.
- An activity function must be triggered by an activity trigger.

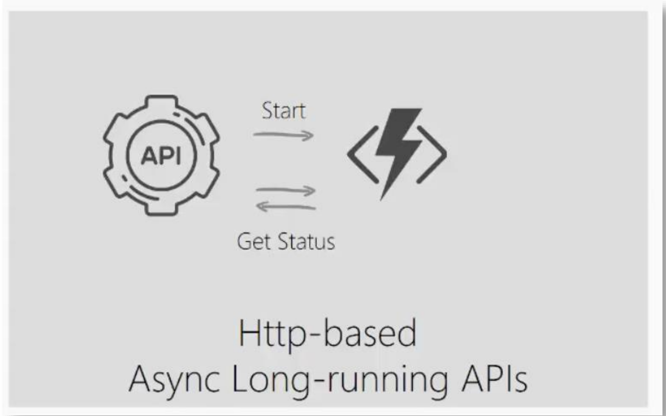
What can you do with Durable Functions?

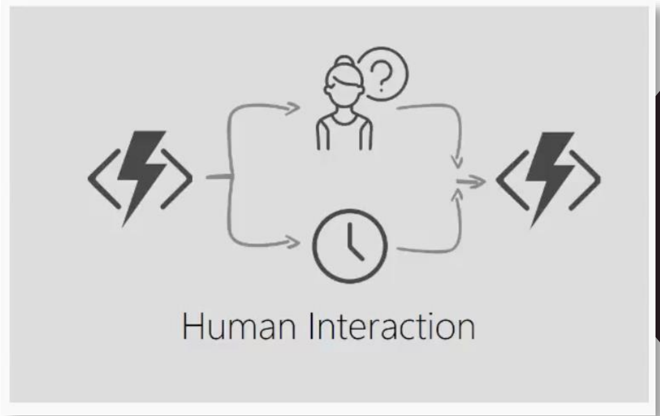




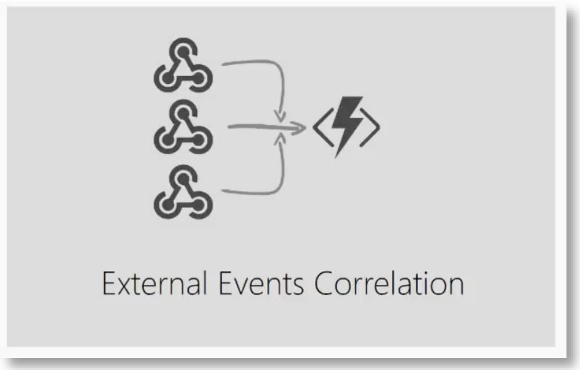








Durable Entities



Based on Azure Functions, with special trigger

Expose operations for reading and updating internal state or interact with other entities

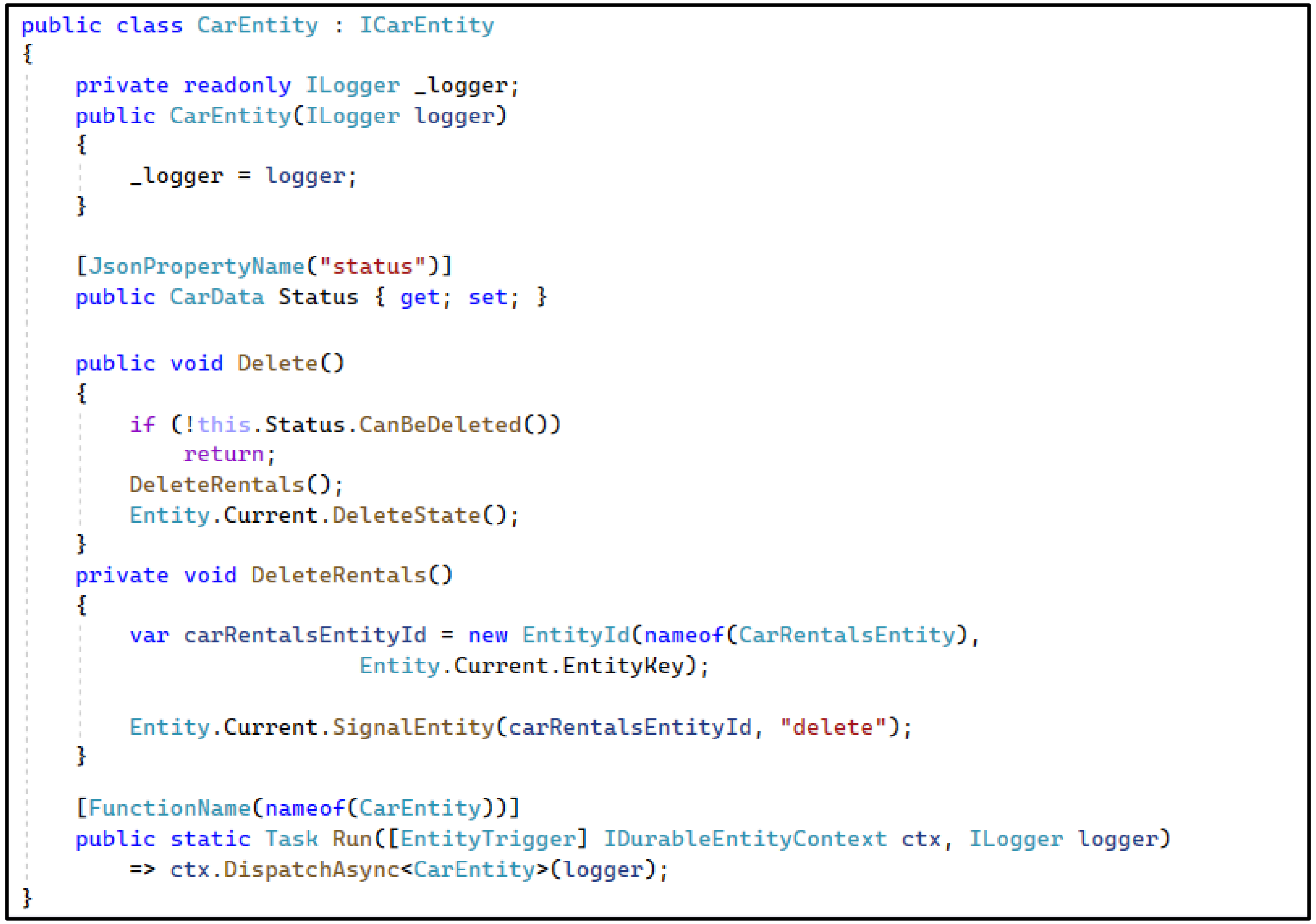
Accessible via Entity ID composed by:

- Entity Name
- Entity Key

Every operation can be accessed using:

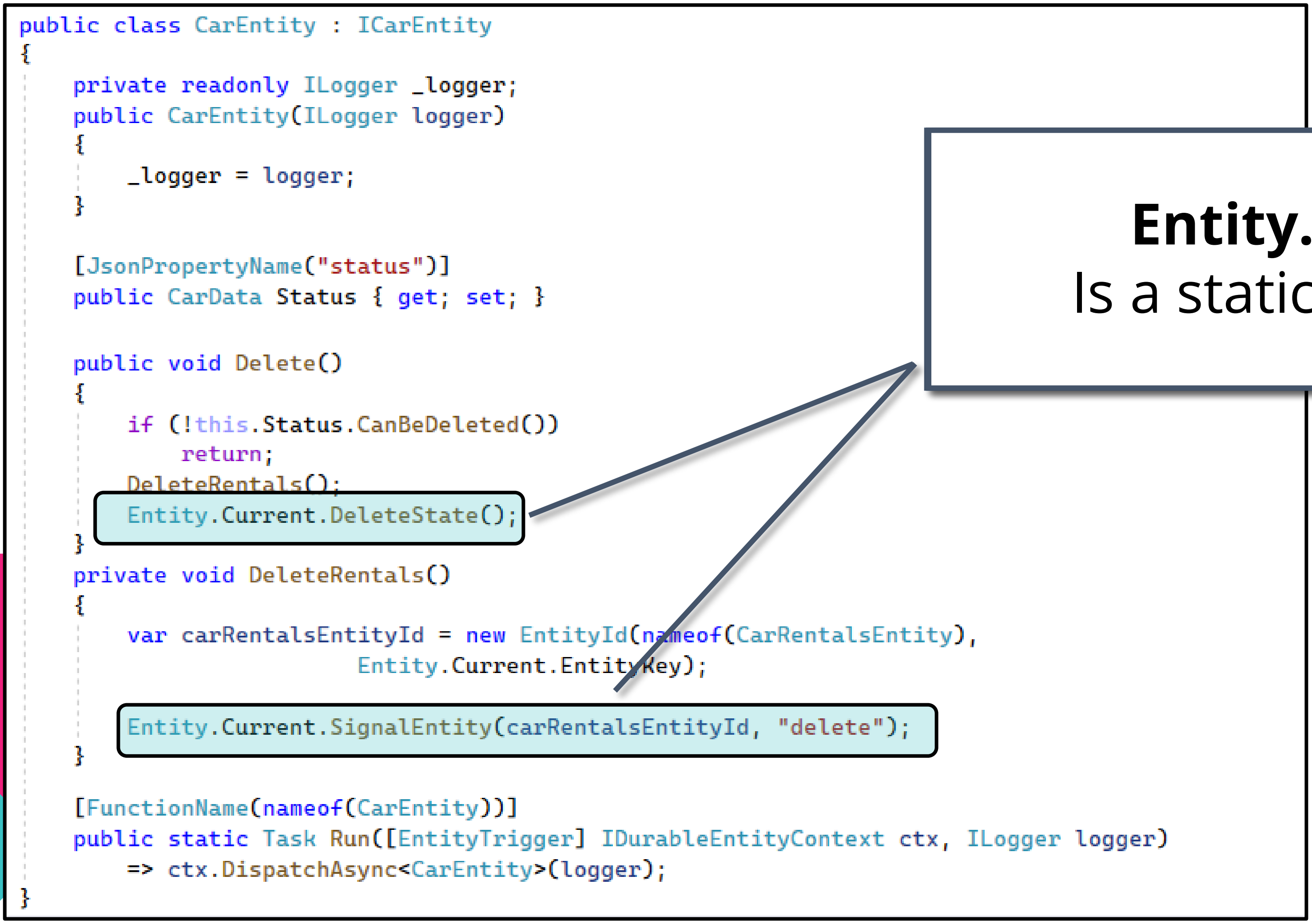
- Entity ID
- Operation Name
- Operation Input
- Scheduled time*

The Entity.Current context!!



```
public class CarEntity : ICarEntity
{
    private readonly ILogger _logger;
    public CarEntity(ILogger logger)
    {
        _logger = logger;
    }

    [JsonPropertyName("status")]
    public CarData Status { get; set; }

    public void Delete()
    {
        if (!this.Status.CanBeDeleted())
            return;
        DeleteRentals();
        Entity.Current.DeleteState();
    }
    private void DeleteRentals()
    {
        var carRentalsEntityId = new EntityId(nameof(CarRentalsEntity),
                         Entity.Current.EntityKey);

        Entity.Current.SignalEntity(carRentalsEntityId, "delete");
    }

    [FunctionName(nameof(CarEntity))]
    public static Task Run([EntityTrigger] IDurableEntityContext ctx, ILogger logger)
        => ctx.DispatchAsync<CarEntity>(logger);
}
```

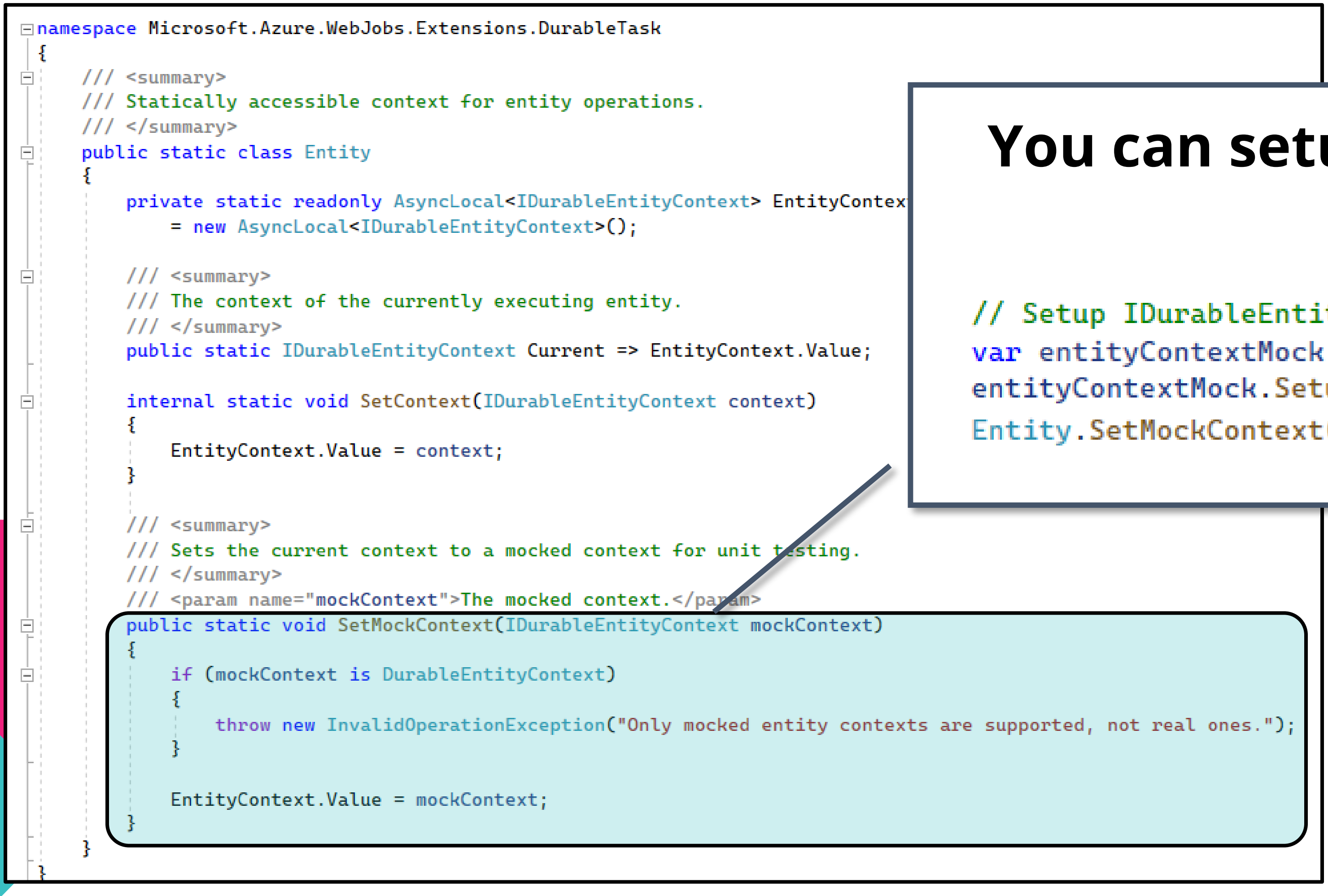
The Entity.Current context!!



```
public class CarEntity : ICarEntity
{
    private readonly ILogger _logger;
    public CarEntity(ILogger logger)
    {
        _logger = logger;
    }

    [JsonPropertyName("status")]
    public CarData Status { get; set; }

    public void Delete()
    {
        if (!this.Status.CanBeDeleted())
            return;
        DeleteRentals();
        Entity.Current.DeleteState();
    }
    private void DeleteRentals()
    {
        var carRentalsEntityId = new EntityId(nameof(CarRentalsEntity),
                                 Entity.Current.EntityKey);

        Entity.Current.SignalEntity(carRentalsEntityId, "delete");
    }

    [FunctionName(nameof(CarEntity))]
    public static Task Run([EntityTrigger] IDurableEntityContext ctx, ILogger logger)
        => ctx.DispatchAsync<CarEntity>(logger);
}
```

Entity.Current
Is a static property!!

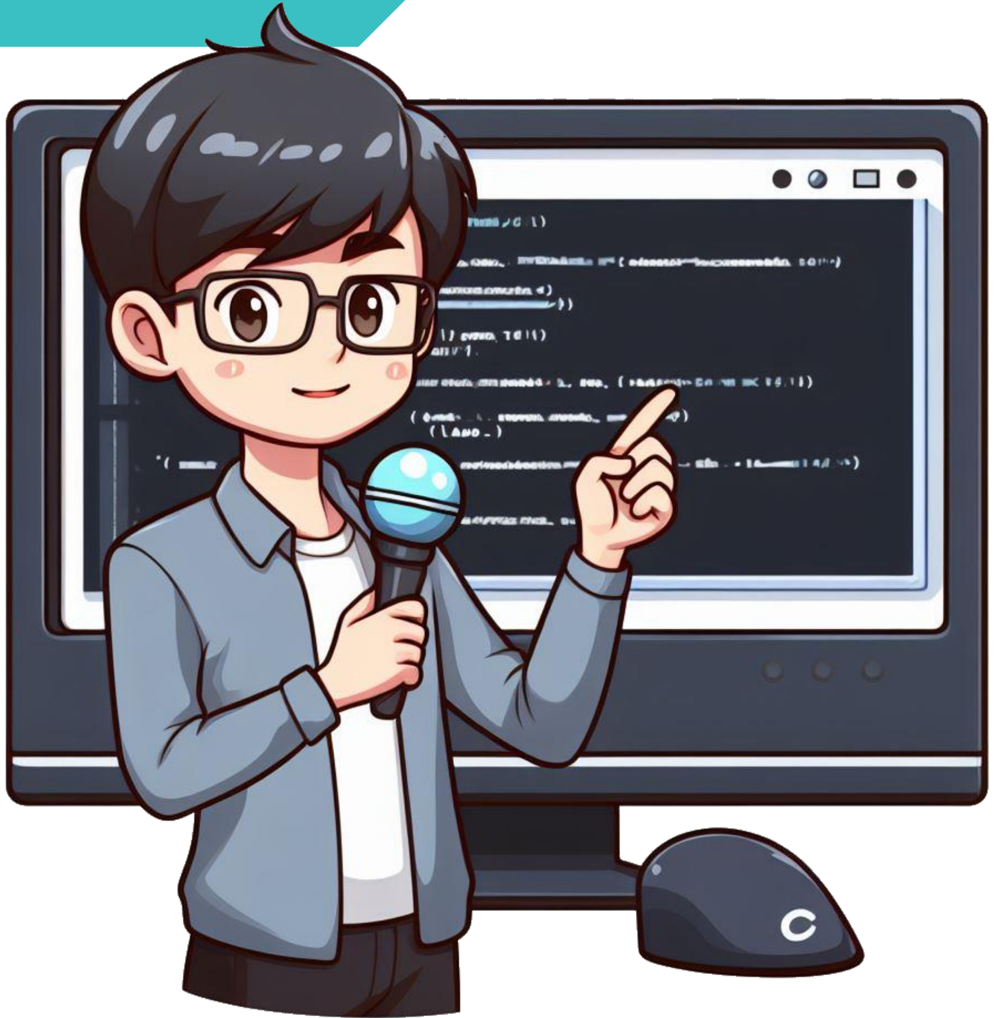
Setting the Entity.Current context!!



```
namespace Microsoft.Azure.WebJobs.Extensions.DurableTask
{
    /// <summary>
    /// Statically accessible context for entity operations.
    /// </summary>
    public static class Entity
    {
        private static readonly AsyncLocal<IDurableEntityContext> EntityContex
            = new AsyncLocal<IDurableEntityContext>();

        /// <summary>
        /// The context of the currently executing entity.
        /// </summary>
        public static IDurableEntityContext Current => EntityContext.Value;

        internal static void SetContext(IDurableEntityContext context)
        {
            EntityContext.Value = context;
        }

        /// <summary>
        /// Sets the current context to a mocked context for unit testing.
        /// </summary>
        /// <param name="mockContext">The mocked context.</param>
        public static void SetMockContext(IDurableEntityContext mockContext)
        {
            if (mockContext is DurableEntityContext)
            {
                throw new InvalidOperationException("Only mocked entity contexts are supported, not real ones.");
            }

            EntityContext.Value = mockContext;
        }
    }
}
```

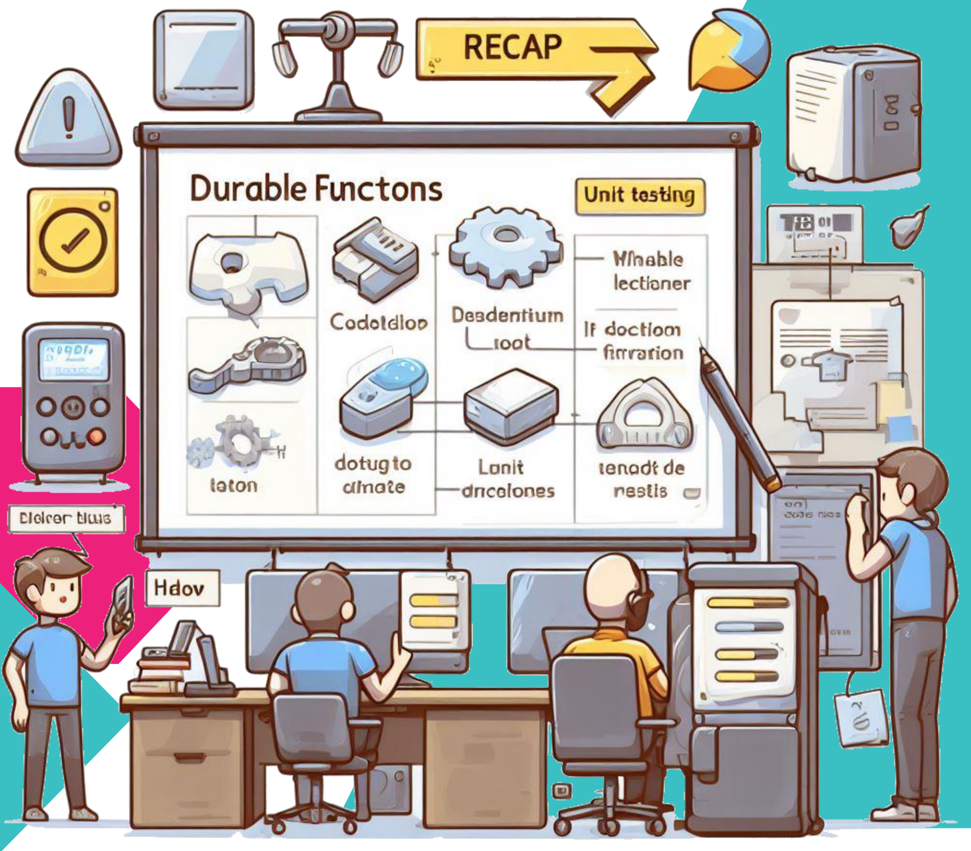
You can setup the context during the test phase!

```
// Setup IDurableEntityContext
var entityContextMock = new Mock<IDurableEntityContext>();
entityContextMock.SetupGet(e => e.EntityKey).Returns(carPlate);
Entity.SetMockContext(entityContextMock.Object);
```



Testing a durable function!!!

Takeaway



Avoid static reference

If create a custom trigger or binding use interface or POCO class for the payload

Dependency Injection allows you to avoid custom Bindings and make function testable

Put business logic in external classes and use functions for the flow

Thanks for the attention !!

Massimo Bonanni

massimo.bonanni@microsoft.com
@massimobonanni

aka.ms/maxlinkedin

References

- Azure Functions Documentation
- Azure Functions Code Samples
- Azure Functions - Unit and Integration Testing
- GitHub Demo - ServerlessCarRent

Grazie!!!