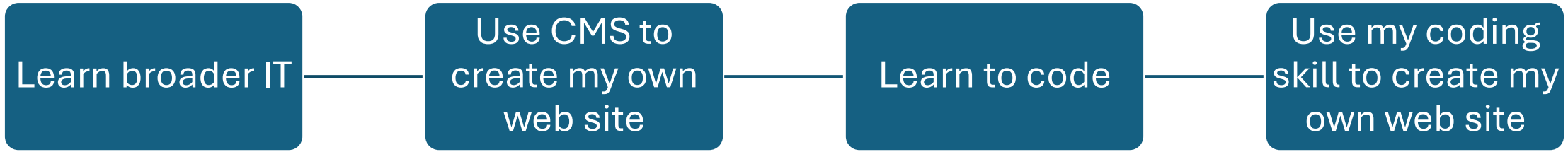
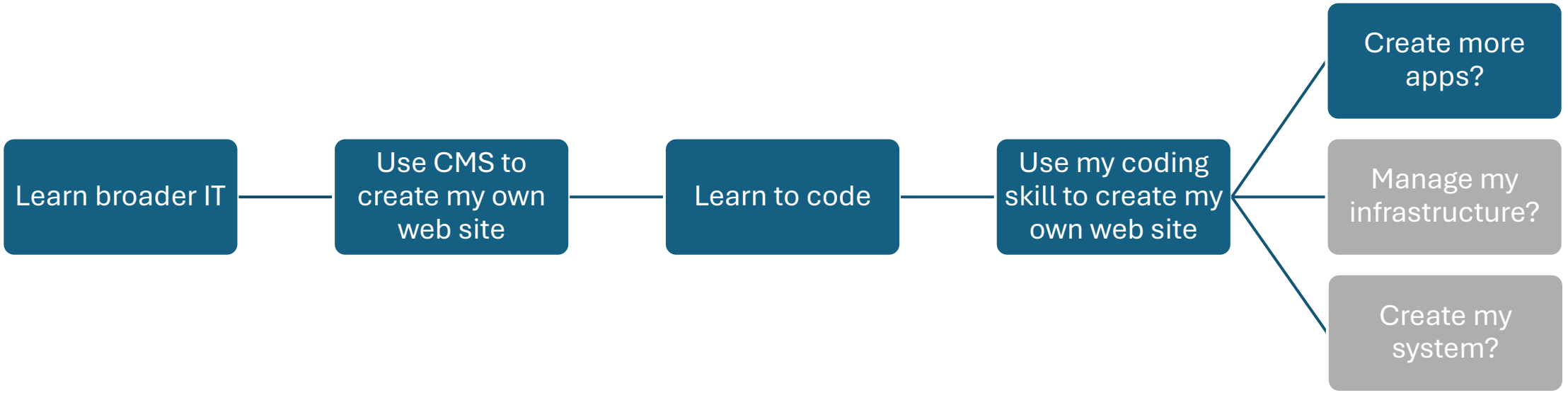
Why and how build your own web server

In C#

My (pretty) common developer journey



What next


Learn broader IT
Use CMS to create my own web site
Learn to code
Use my coding skill to create my own web site
Create more apps?
Manage my infrastructure?
Create my system?

SRE Keys difference with SWE

ASPECT	SRE	SWE
Primary Focus	System	Application
Key Goal	Minimize downtime & incidents	Build new applications/features
Main Work	Automation, monitoring, operational management	Writing business logic & product code

Why build a system and a web server

- Great opportunity to learn about fundamental service
- Help to understood better how actual services works
- Can show your software engineer skill
- Server web is simple system but with variety of complexes problem

Before how, the what

- Functionalities of a web server
  - Service running on a host
  - Listening on a port
  - Manage HTTP requests
  - Generate HTTP responses
- Non-functionalities of a web server
  - Secure with authentication
  - Reliable (Should not crash when there is heavy usage)
  - Performance and scalability
  - Easy to operate

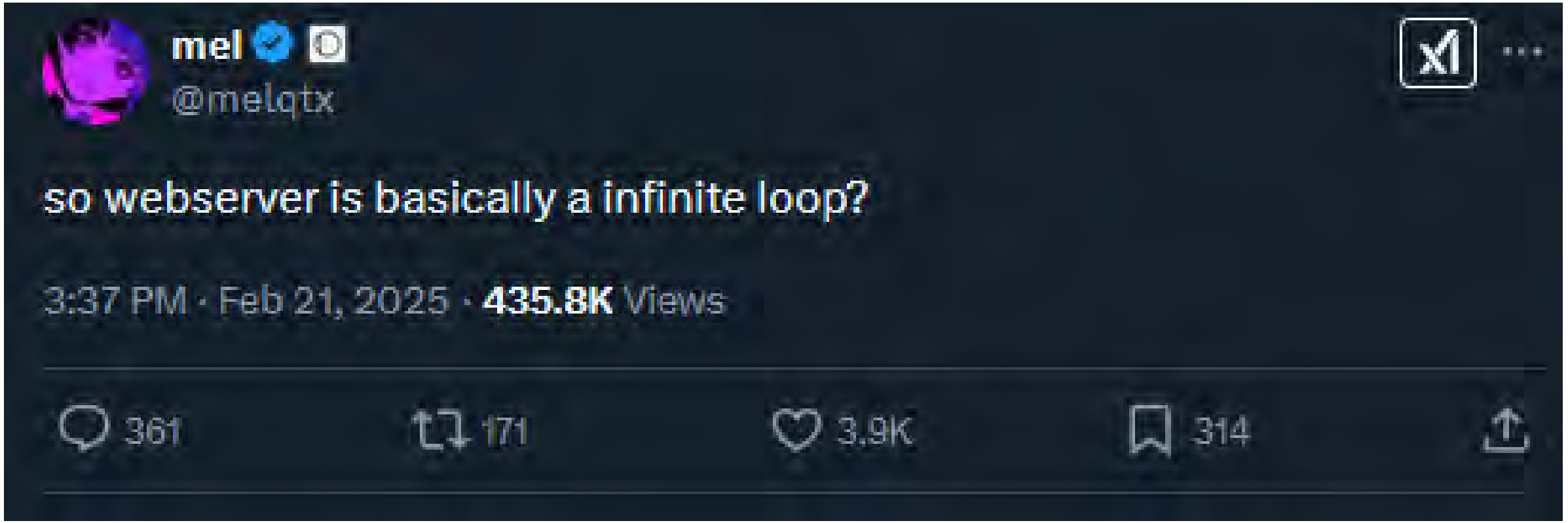
First challenge

- Multiplatform web server working on Windows and Linux
  - Today .Net core can be execute in Windows and Linux
  - Platform related topic as: Path, Metric, Scripting, can be handle by propre abstraction

Step 1: Create the server and connect to it

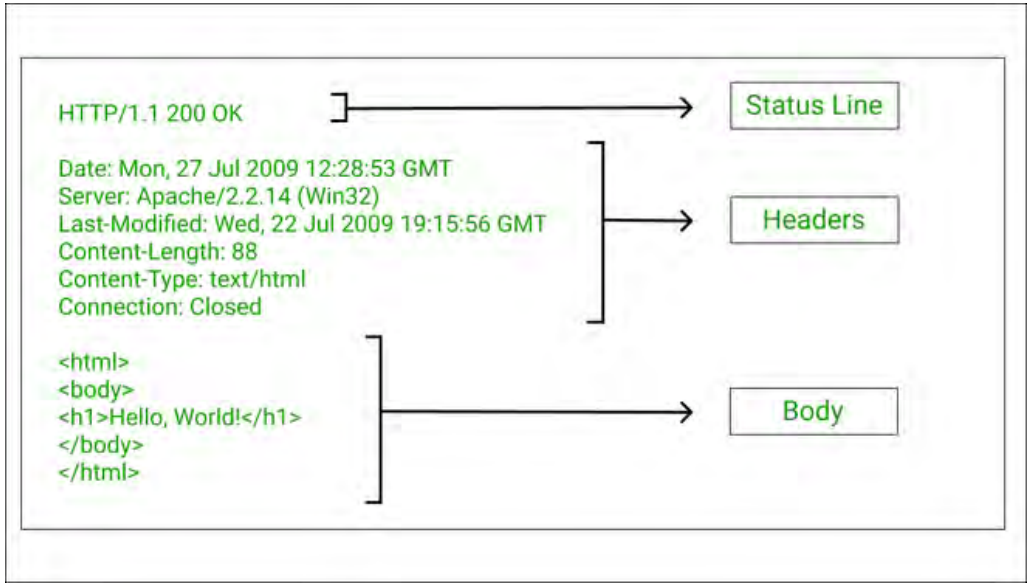
```
TcpListener ServerListener = new TcpListener(IPAddress.Any, 8080);
ServerListener.Start();
while (true)
{
    if (ServerListener.Pending())
    {
        Socket client = ServerListener.AcceptSocket();
        Console.WriteLine("Client connected");
    }
}
```

Step 1: Create the server and connect to it



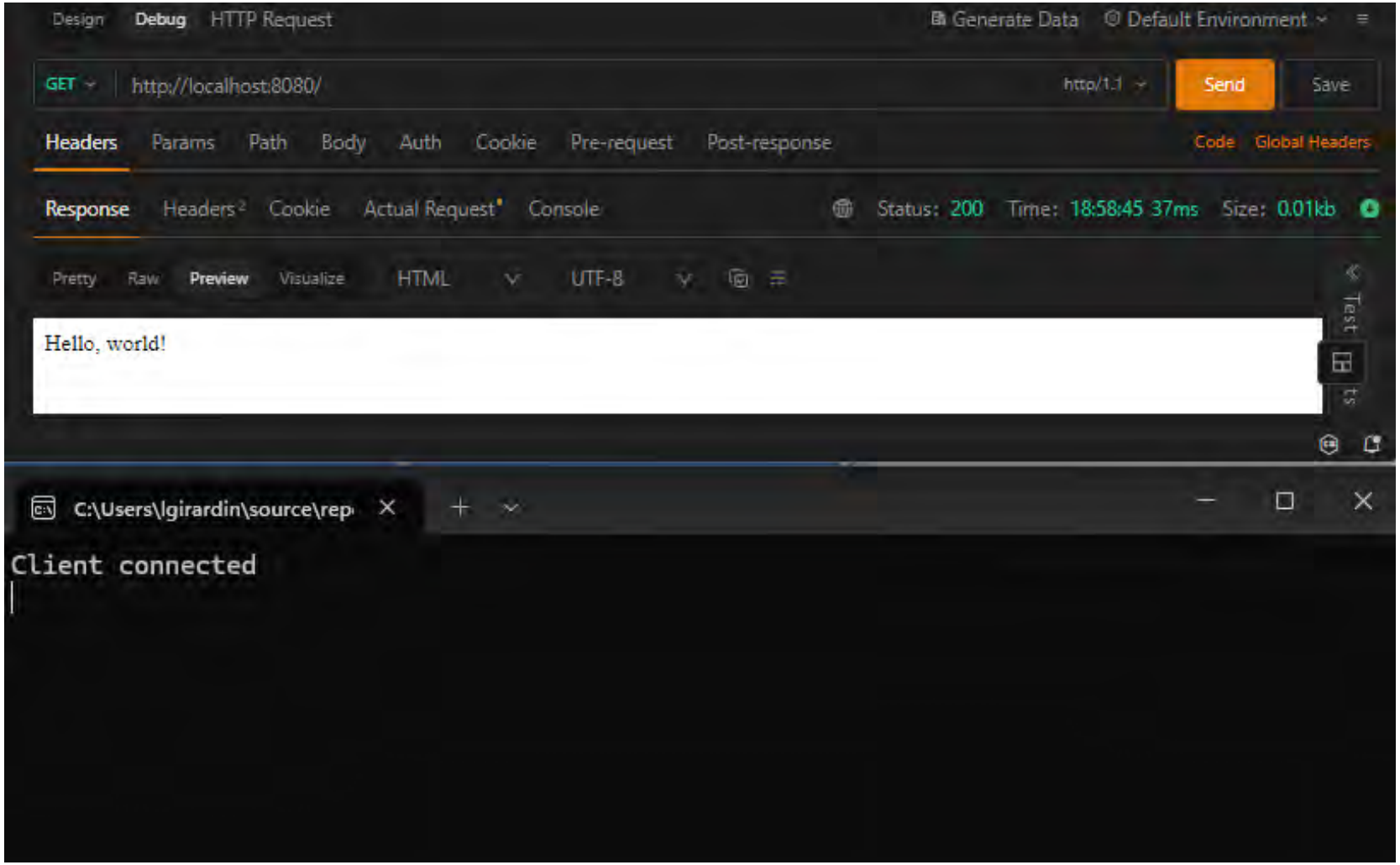
Step 2: Generate the simplest response

https://www.geeksforgeeks.org/state-the-core-components-of-an-http-response/




```
while (true)
{
    if (ServerListener.Pending())
    {
        Socket client = ServerListener.AcceptSocket();
        Console.WriteLine("Client connected");

        string response = "HTTP/1.1 200 OK\r\n" +
                          "Content-Length: 13\r\n" +
                          "Content-Type: text/plain\r\n" +
                          "\r\n" +
                          "Hello, world!";

        byte[] msg = Encoding.UTF8.GetBytes(response);
        await client.SendAsync(msg, SocketFlags.None);

    }
}
```

Step 2: Generate the simplest response

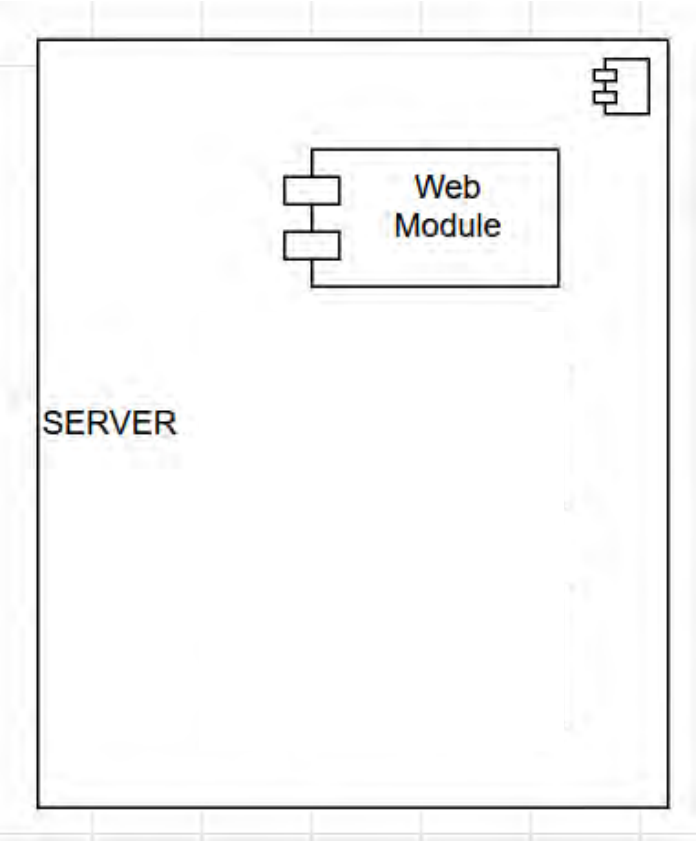


Step 3 – Do the same with the HTTP Request

Challenge number 2: Handling connection

- Requirement
  - Must use a modular architecture
  - Connections must be accepted or not
  - Lifecycle of the connection must be managed
  - The HTTP request must be served and return an HTTP Response

The web module – Manage the connection




```
public async Task RunAsync()
{
    TcpListener listener = connectionManager.Listener;
    while (true)
    {
        if (listener.Pending())
        {
            ISocketWrapper client = new SocketWrapper(listener.AcceptSocket());
            await connectionManager.AcceptConnectionsAsync(client);
        }
    }
}
```

The web module – Accept the connection

```
public async Task AcceptConnectionsAsync(ISocketWrapper client)
{

    Guid clientId = Guid.NewGuid();

    if (_clients.Count < ServerConfiguration.Instance.MaxAllowedConnections)
    {
        _clients.TryAdd(clientId, client);
        ExecutionContext.Current.Logger.LogInfo($"Client connected");

        // Handle client in a separate thread
        TaskFactory.StartNew(() => HandleClientAsync(client, clientId));
    }
    else if (_clientsRejected.Count < ServerConfiguration.Instance.MaxRejectedConnections)
    {
        _clientsRejected.TryAdd(clientId, client);

        TaskFactory.StartNew(() => HandleClientRejectAsync(client, clientId));
    }
    else
    {
        ExecutionContext.Current.Logger.LogError("Too many connection");
        client.Dispose();
        return;
     }
}
```

The Web Module – Shorter path, reject connection

```
private async Task HandleClientRejectAsync(ISocketWrapper client, Guid clientId)
{
    HttpResponse errorResponse = await ErrorHandler.HandleErrorAsync(new
TooManyConnectionsException());
    await ConnectionUtils.SendResponseAsync(client, errorResponse, null);

     _clientsRejected.TryRemove(clientId, out client);

     HttpResponse errorResponse = new HttpResponse(500)
     {
         Body = new MemoryStream(Encoding.UTF8.GetBytes(ex.Message))
     };
     await ConnectionUtils.SendResponseAsync(client, errorResponse, null);
     client.Dispose();
}
```

The Web Module – Accept the request

ConnectionManager have specific attribute name pipeline

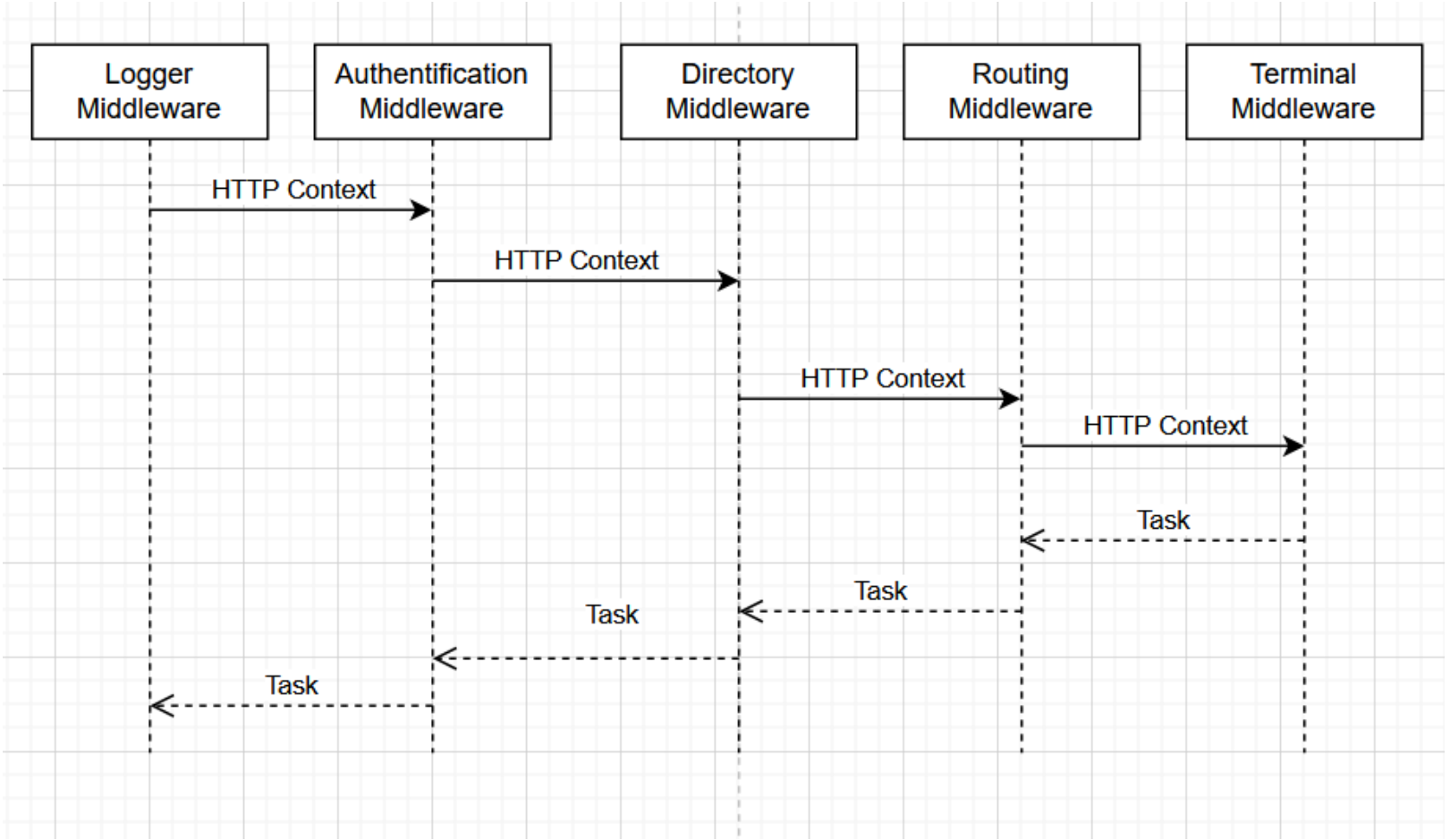
```
private Func<HttpContext, Task> _pipeline;
```

```
private async Task HandleClientAsync(ISocketWrapper client, Guid clientId)
{
    //Parsing request
    var taskParsing = TaskFactory.StartNew(() => ConnectionUtils.ParseRequestAsync(client)).Unwrap();
    Request request = await taskParsing;
    HttpContext context = new HttpContext(request, null);

    //Execute request
    var taskPipeline = TaskFactory.StartNew(() => _pipeline(context)).Unwrap();
    await taskPipeline;

    //Sending response
    var taskSending = TaskFactory.StartNew(() => ConnectionUtils.SendResponseAsync(client,
context.Response, context.Request)).Unwrap();
    await taskSending;
}
```

The Web Module – What is a pipeline

- When a connection arrive, we have many action to perform on the request or on the response
  - Log it
  - Check the authorization or authentication
  - Route the request
  - Etc…
- Each action is done by a middleware
- The pipeline, is the chain of execution of these middleware

The Web Module – Pipeline example



The Web Module- Pipeline configuration

```
public void Init()
{
  //Create pipeline
  TerminalMiddleware terminalMiddleware = new TerminalMiddleware();
  RoutingMiddleware routingMiddleware = new RoutingMiddleware(terminalMiddleware.InvokeAsync);
  DirectoryMiddleware directoryMiddleware = new DirectoryMiddleware(routingMiddleware.InvokeAsync);
  AuthMiddleware authMiddleware = new AuthMiddleware(directoryMiddleware.InvokeAsync);
  LoggerMiddleware loggerMiddleware = new LoggerMiddleware(authMiddleware.InvokeAsync);

  ConnectionManager webModule = new ConnectionManager(loggerMiddleware.InvokeAsync);
}
```

The Web Module – Example of a middleware (Router)

```
public async Task InvokeAsync(HttpContext context)
{

    if (context.Request.Verb.ToUpper().Equals("GET"))
    {
        await GetHandlerAsync(context);
    }
    else
    {
        context.Response = new HttpResponse(404);
    }

    await _next(context);

}
```

The Web Module – Example of a middleware (Router)

```
private async Task GetHandlerAsync(HttpContext context)
{
    Request request = context.Request;
    string mainDirectory = ServerConfiguration.Instance.RootDirectory;
    string filePath = Path.Combine(mainDirectory, Utils.GetFilePath(request.Path));

    if (_FileManager.Exists(filePath))
    {
        string fileExtension = Path.GetExtension(filePath).Substring(1).ToLowerInvariant();
        string mimeType = fileExtension switch
        {
            // Images
            "jpg" or "jpeg" => "image/jpeg",
            "png" => "image/png",

            // Documents textuels
            "txt" => "text/plain",
            "html" or "htm" => "text/html",
            "css" => "text/css",
            "js" => "application/javascript",

            _ => null
        };
        if (mimeType != null)
        {
            byte[] data = await File.ReadAllBytesAsync(filePath);
            HttpResponse httpResponse = new HttpResponse(200)
            {
                Body = new MemoryStream(data)
            };
            httpResponse.AddHeader("Content-Type", mimeType);
            httpResponse.AddHeader("Content-Length", httpResponse.Body.Length.ToString());
            context.Response = httpResponse;
        }
    }
    else
    {
        context.Response = new HttpResponse(404);
    }
}
```

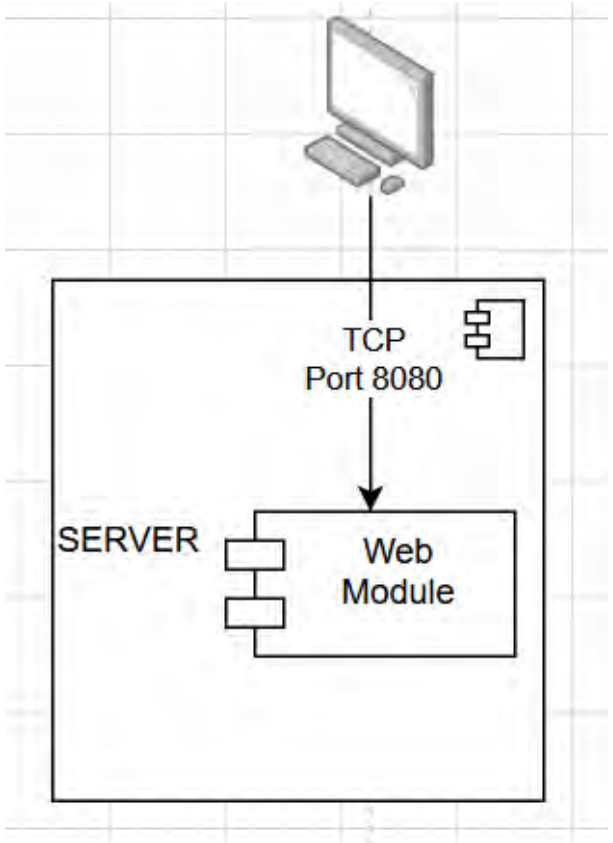
Middleware architecture advantage

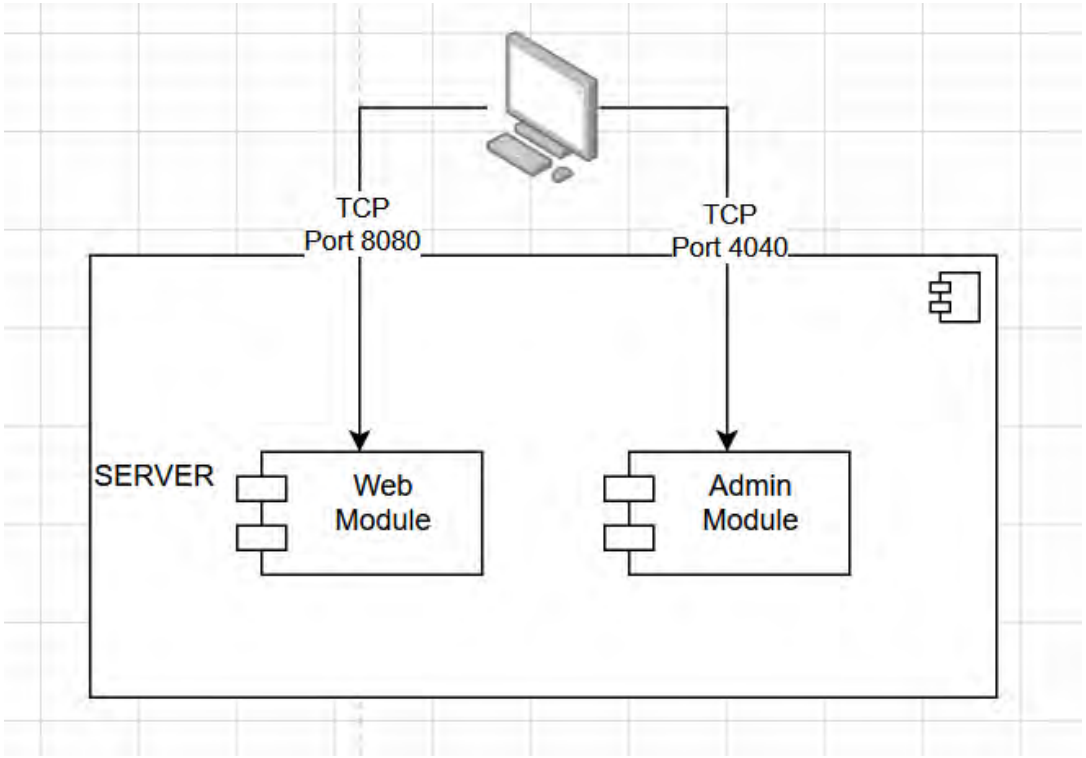
- **Modular & Reusable**: Simplifies development and maintenance.
- **Extensible**: Easily add or remove features.
- **Maintainable**: Improves debugging and testing.
- **Secure**: Centralized authentication and rate limiting.

Challenge number 3 : Administrate and operate a modern web server

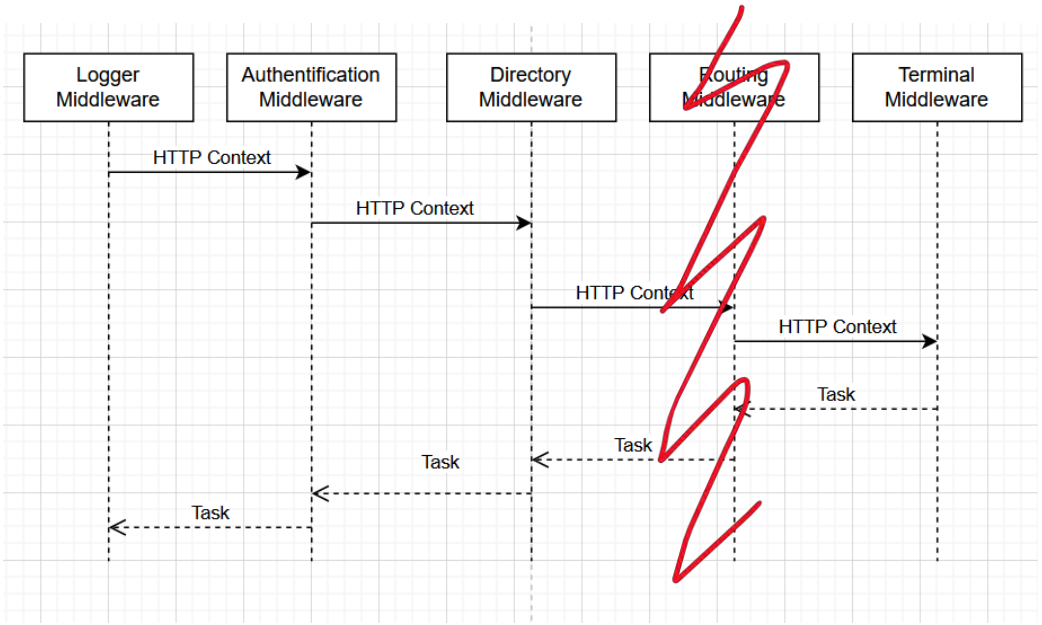
- Requirement
  - Allow to perform operation remotely on the service by an administrator
  - Must be HTTP endpoint
  - Required authentication
  - Must be independent of the first module

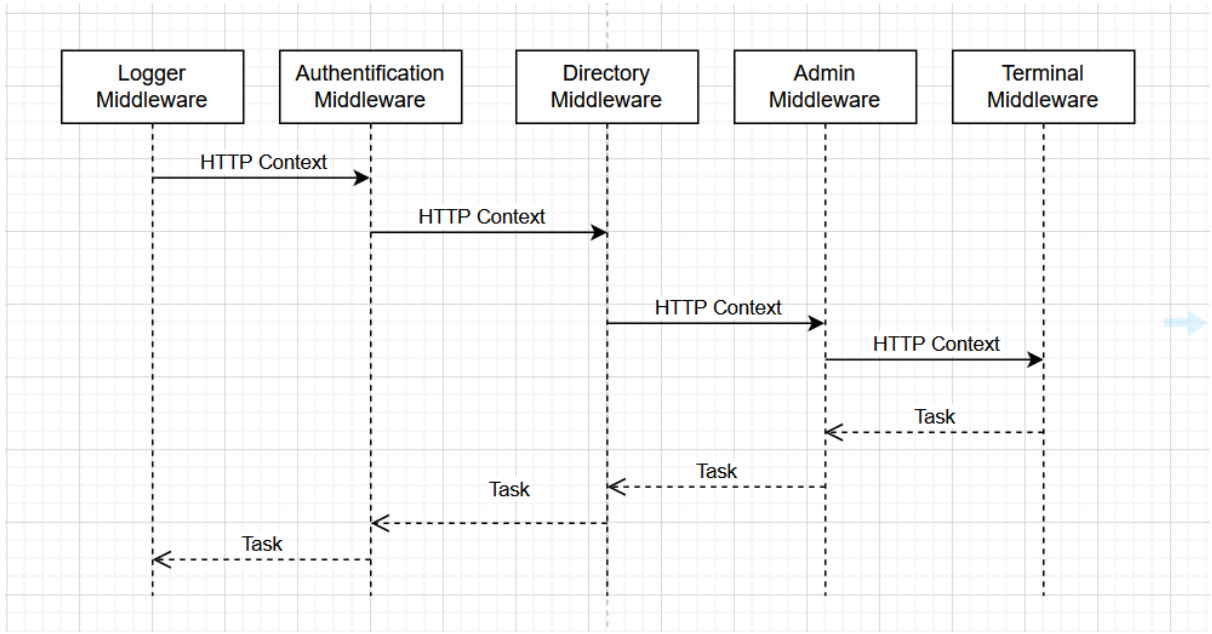
As-is => To be


TCP
Port 8080
SERVER
Web
Module


TCP
Port 8080
TCP
Port 4040
SERVER
Web
Module
Admin
Module

Admin module – Pipeline modification





The Admin Module- Pipeline configuration

```
public void init()
{
    TerminalMiddleware terminalMiddleware = new TerminalMiddleware();
    AdminMiddleware adminMiddleware = new AdminMiddleware(terminalMiddleware.InvokeAsync);
    DirectoryMiddleware adminDirectoryMiddleware = new
DirectoryMiddleware(adminMiddleware.InvokeAsync);
    AuthMiddleware adminAuthenticationMiddleware = new
AuthMiddleware(adminDirectoryMiddleware.InvokeAsync);
    LoggerMiddleware adminLoggerMiddleware = new LoggerMiddleware(adminAuthMiddleware.InvokeAsync);

  ConnectionManager admin = new ConnectionManager(adminLoggerMiddleware.InvokeAsync)
}
```

The Web Module – Example of a middleware (Admin)

```
if (request.Path.ToLower().Equals(adminURL + "/status"))
{
    response = new HttpResponse(200)
    {
        Body = new MemoryStream(Encoding.UTF8.GetBytes(Server.Instance.Status))
    };
}
else if (request.Path.ToLower().Equals(adminURL + "/reload"))
{
    ServerConfiguration.Instance.ReloadConfiguration();
    Response = new HttpResponse(200)
    {
        Body = new MemoryStream(Encoding.UTF8.GetBytes("Configuration reloaded"))
    };

}
else if (request.Path.ToLower().Equals(adminURL + "/restart"))
{
    Task.Run(() => Server.Instance.StoppAsync(true));
    response = new HttpResponse(200)
    {
        Body = new MemoryStream(Encoding.UTF8.GetBytes("Restarting...."))
    };
}
else if (request.Path.ToLower().Equals(adminURL + "/logs"))
{
    StringBuilder logs = new StringBuilder();
    foreach (var logEntry in ((Logger)Logger.Instance).GetLogHistory())
    {
        logs.Append(logEntry.ToString() + "\n");
    }
    response = new HttpResponse(200)
    {
        Body = new MemoryStream(Encoding.UTF8.GetBytes(logs.ToString()))
    };
}
```

The Web Module – Example of a middleware (Auth)

```
public async Task InvokeAsync(HttpContext context)
{
    if (context.Request.Headers.ContainsKey("Authorization"))
    {
        string authHeader = context.Request.Headers["Authorization"];
        string credentials =
Encoding.UTF8.GetString(Convert.FromBase64String(authHeader.Substring("Basic ".Length).Trim()));
        string[] credentialParts = credentials.Split(':');
        if (credentialParts.Length == 2)
        {
            string username = credentialParts[0];
            string password = credentialParts[1];

            string mainDirectory = ServerConfiguration.Instance.RootDirectory;
            string filePath = Path.Combine(mainDirectory, Utils.GetFilePath(context.Request.Path));
            DirectoryConfig directoryConfig = ServerConfiguration.Instance.GetDirectory(filePath);
            if (IsValidUser(directoryConfig, username, password))
            {
                context.isAuth = true;
            }
        }
    }

    await _next(context);
}
```

Admin Module – How anticipate all admin use case in endpoint

- We can create endpoint for the most generic case (Stop, restart, status, log, change configuration…)
- We cannot be exhaustive; we need to offer flexibility for administrator

Admin module – Script execution

```
else if (request.Path.ToLower().StartsWith(adminURL + "/script"))
{
    if (ServerConfiguration.Instance.AdminScript)
    {
        string scriptsDirectory = ServerConfiguration.Instance.AdminDirectory.Path;

        if (request.queryParameters.Count > 0)
        {

            response = RunScript(scriptsDirectory, request.queryParameters);
        }
        else
        {
            string scriptExtension = string.Empty;
            if (ServerConfiguration.Instance.Platform.Equals("WIN"))
                scriptExtension = ".ps1";
            else
                scriptExtension = ".sh";

            string bodyScript = string.Empty;
            foreach (string script in Directory.GetFiles(scriptsDirectory))
            {
                FileInfo fileInfo = new FileInfo(script);
                if (fileInfo.Extension.Equals(scriptExtension))
                    bodyScript += $"{fileInfo.Name}\n";
            }

            response = new HttpResponse(200)
            {
                Body = new MemoryStream(Encoding.UTF8.GetBytes(bodyScript))
            };
        }
    }
}
```

Challenge number 4 – Effective logging

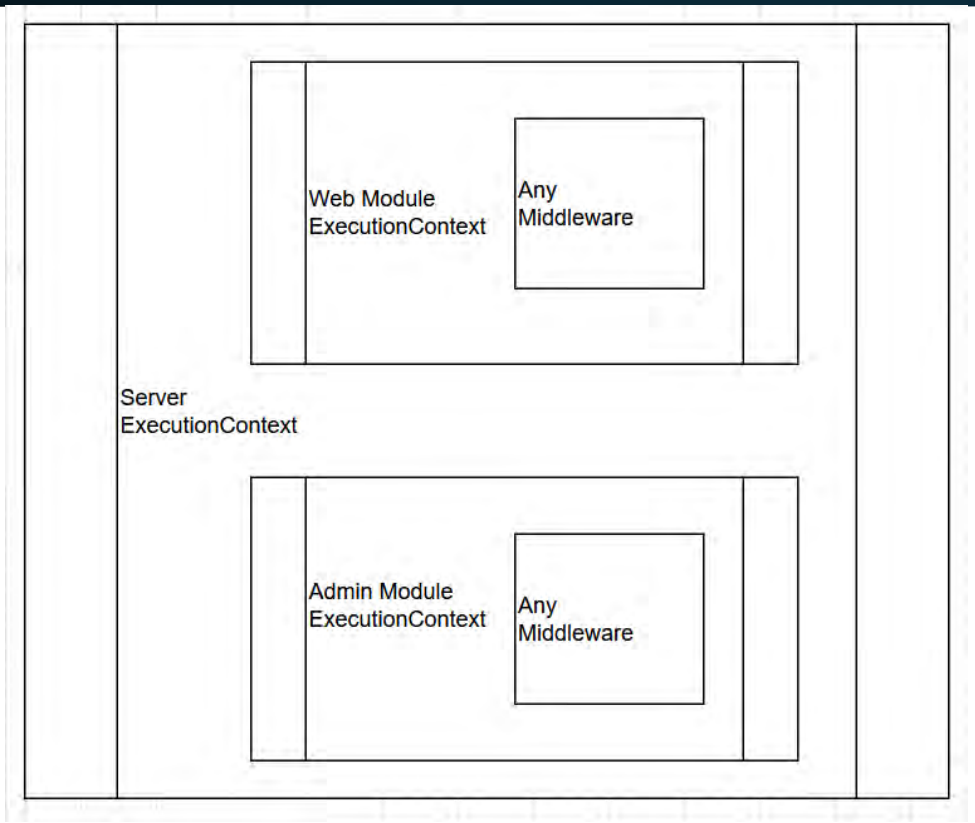
- Log are very important for SRE
- Requirement:
  - Log must have different level
  - Log configuration must be independent between module
- Example of nice log
  - Date – Module – Thread – Level - message

```
2/21/2025 8:35:49 PM - Module: Server     - Thread ID: 1  - INFO       - PandApache3 is starting
2/21/2025 8:35:49 PM - Module: Telemetry  - Thread ID: 1  - INFO       - Starting Telemetry module
2/21/2025 8:35:50 PM - Module: Web        - Thread ID: 1  - INFO       - Starting Connection manager module
2/21/2025 8:35:50 PM - Module: Web        - Thread ID: 1  - INFO       - Web server listening on 127.0.0.1:8080
2/21/2025 8:35:50 PM - Module: Admin      - Thread ID: 1  - INFO       - Starting Connection manager module
2/21/2025 8:35:50 PM - Module: Admin      - Thread ID: 1  - INFO       - Admin server listening on 127.0.0.1:4040
2/21/2025 8:35:50 PM - Module: Server     - Thread ID: 1  - INFO       - PandApache3 is up and running!
2/21/2025 8:35:50 PM - Module: Telemetry  - Thread ID: 7  - INFO       - Running Telemetry module
2/21/2025 8:35:50 PM - Module: Web        - Thread ID: 10 - INFO       - Running Connection manager module
Error rotating log file: Cannot create a file when that file already exists.
2/21/2025 8:35:50 PM - Module: Admin      - Thread ID: 9  - INFO       - Running Connection manager module
```

The logging issue

- The web and admin module are 2 instances of the same class
- Both can call the class LoggerMiddleware that wil generate a log
- How do I know with module call it ?

Understood the context execution




```
public async Task RunAsync()
{
    Status = "PandApache3 is up and running!";

    List<Task> tasks = new List<Task>();
    foreach (var moduleName in Modules.Keys)
    {
        tasks.Add(Task.Run(() => Modules[moduleName].RunAsync()));
    }

    await Task.WhenAll(tasks);
}
```

- The Server is the main execution context of the code
- When a new task is run, we enter in a child execution context
- The server execute each module in independent Task

ExecutionContext class

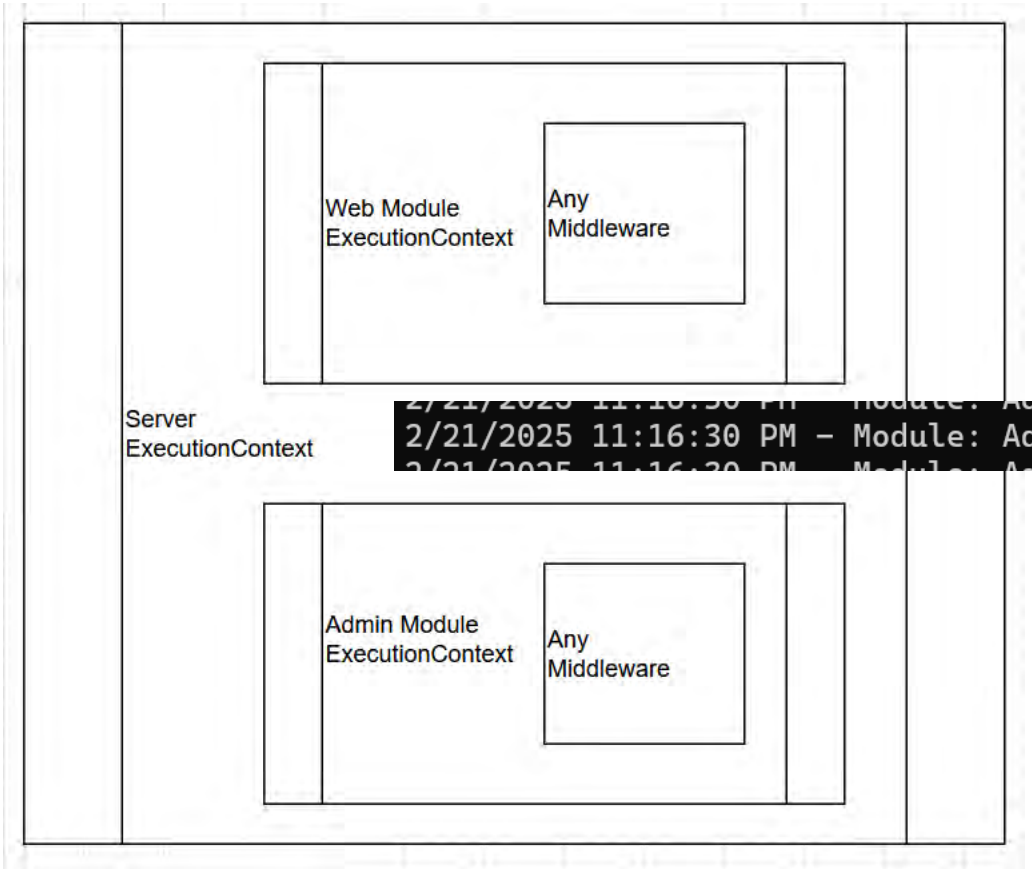
```
public static class ExecutionContext
{
    private static AsyncLocal<VLogger> _logger = new AsyncLocal<VLogger>();

    public static VLogger Logger
    {
        get => _logger.Value;
        set => _logger.Value = value;
    }
}
```

Starting a module

```
public async Task StartAsync()
{
    ExecutionContext.Logger = new VLogger(ModuleName);
    ExecutionContext.Logger.LogInfo("Starting Connection manager module");
    //Continue to start the module
}
```

Here we change the variable Logger for this Execution context, and all future children

ExecutionContext usage




```
ExecutionContext.Logger.LogInfo("LoggerMiddleware invoked");
```

```
2/21/2025 11:16:30 PM - Module: Admin      - Thread ID: 17 - INFO       - LoggerMiddleware invoked
```

Many other challenge incoming

- Performance challenge – caching, multiplex request…
- Security – With HTTPS, Oauth…
- Monitoring – Generate the metric, new endpoint…

Bonus challenge

- What name for your Server Web ?

Bonus challenge

https://github.com/MarieLePanda/PandApache3

Thank you