**What is Middleware in ASP.NET Core**  
In ASP.NET Core, Middleware is a piece of software that can handle an HTTP request or response. A given middleware component in ASP.NET Core has a very specific purpose. For example we may have a middleware component that authenticates a user, another piece of middleware to handle errors, yet another middleware to serve static files such as JavaScript files, CSS files, Images etc.    
  
  
It is these middleware components that we use to setup a request processing pipeline in ASP.NET Core. It is this pipeline that determines how a request is processed. The request pipeline is configured as part of the application startup by the Configure()method in Startup.cs file.  The following is the code in Configure() method.

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

    if (env.IsDevelopment())

    {

        app.UseDeveloperExceptionPage();

    }

    app.Run(async (context) =>

    {

        await context.Response.WriteAsync("Hello World!");

    });

}

As you can see, the code in the Configure() method that is generated by the empty project template sets up a **very simple request processing pipeline**with just two pieces of middleware.   
  
UseDeveloperExceptionPage is one middleware and the second middleware is setup using the Run() method. As it stands right now with this very simple request processing pipeline, all our application can do is write a message to the response object that will be displayed by the browser. We will come back and understand the code in this method in detail later. For now let’s understand what is middleware and how it works in asp.net core.   
  
The following diagram helps us understand middleware components and how they fit in a request processing pipeline   
  
   
  
In ASP.NET Core, **a Middleware component has access to both - the incoming request and the outgoing response.** So a Middleware component may process an incoming request and pass that request to the next piece of middleware in the pipeline for further processing. For example, if you have a **logging middleware**, it might simply log the time the request is made and pass the request to the next piece of middleware for further processing.  
  
**A middleware component may handle the request and decide not to call the next middleware in the pipeline.** This is called short-circuiting the request pipeline. Short-circuiting is often desirable because it avoids unnecessary work. For example, if the request is for a static file like an image or css file, the StaticFiles middleware can handle and serve that request and short-circuit the rest of the pipeline. This means in our case, the StaticFiles middleware will not call the MVC middleware if the request is for a static file.  
  
A middleware component may handle an incoming HTTP request by generating an HTTP response. For example, **mvcmiddleware**in the pipeline handles a request to the URL **/employees**and returns a list of employees. As we progress through this course, in our upcoming sessions we will be including the **mvcmiddleware**in the request processing pipeline of our application.   
  
**A middleware component may also process the outgoing response**. For example, the logging middleware component may log the time the response is sent. In addition it may also calculate the over all time taken to process the request by computing the difference between request received and response sent times.   
  
**Middleware components are executed in the order they are added to the pipeline**. Care should be taken to add the middleware in the right order, otherwise the application may not function as expected. In our upcoming videos, we will discuss with an example, what happens if the middleware components are not added to the processing pipeline in the correct order.   
  
**The middleware components are available as NuGet packages**. This means updates are now handled by NuGet, providing the ability to update each middleware separately.   
  
Depending on your application requirements you may add as many or as few middleware components to the request processing pipeline. For example, if you are developing simple web application with a few static HTML pages and images, then your request processing pipeline may contain just **"StaticFiles" middleware**.   
  
On the other hand, if you are developing a secure data driven web application then you may need several middleware components like StaticFiles middleware, Authentication middleware, Authorization middleware, MVC middleware etc.    
  
The point that I am trying to make is, you have complete control over configuring the request processing pipeline. This also means from a memory and performance standpoint you only pay for the middleware components you have in your request processing pipeline.   
  
Now that we have a basic understanding of what middleware components are and how they fit in a request processing pipeline.