* Introducing .NET Health Checks
  + We are going to use the Microsoft.AspNetCore.Diagnostics.HealthChecks package to add health checks to our application
  + This will allow our front-end to monitor the status of our back-end server
  + This is added easy enough by adding the HealthChecks to our services then adding it to the app (before MapControllerRoute so that the middleware doesn’t look for the health endpoint as a general controller)
  + The default check here simply reports Healthy since we have not added any checks to tell it otherwise
* Adding an ICMP Check
  + Internet Control Message Protocol (ICMP) request check is also known as a PING
  + This checks the presence and availability of a server that should be available in a LAN or WAN
  + One machine sends the ICMP echo request packets to the target host and waits for a reply, reporting the round-trip time
  + We are going to add a health check using these messages with statuses of Healthy, Degraded, and Unhealthy
  + Creating an ICMPHealthCheck Class
    - This is just a class that implements IHealthCheck and its method CheckHealthAsync()
    - This method returns a HealthCheckResult, in our case based on the result of a Ping.SendPingAsync() method call
  + Adding ICMPHealthCheck to the Pipeline
    - To add our own health check to the pipeline, we need to register our class with the HealthChecks middleware
    - This is done with the AddCheck<HealthCheckClass>(“GoodName”) method on the service registration AddHealthChecks() method
  + Improving the ICMPHealthCheck Class
    - We make the host and timeout parameters properties and have them added via ctor
    - We added some better messages that are reported as part of the health checks
  + Updating Middleware Setup
    - Now we need to add these new required parameters to the ICMPHealthCheck constructor
    - We can do this by using the services AddHealthChecks().AddCheck(“GoodName”, new MyHealthCheck())
    - Note that the default messaging behavior will still be to display one message based on the sum of all the health checks
    - We need to change this default behavior
  + Implementing a Custom Output Message
    - To do this we add a new type inheriting from HealthCheckOptions
    - The base type includes a ResponseWriter property that we will set in the ctor to implement our desired behavior
    - We supply an anonymous async method that takes an HttpContext and HealthReport as params and returns a Task
    - We hard-code the status code of the HttpContext to a 200 as the default behavior is to return a 200 if all checks are Healthy and a 503 otherwise
      * We don’t need this now as we are returning a more detailed JSON report
    - We also set the result as a serialization of an anonymous object with a checks element populated from the HealthReport entries
    - We configure the app to use these options by adding it to the application UseHealthChecks(“route”, options) method
* Health Checks in Angular
  + We are now going to build an Angular component that can fetch and display the health check JSON data that we have made available
  + This will require creating
    - The **Component** (\*.ts) file containing the Component class and the module references, functions, variables, etc.
    - The **Template** (\*.html) file written in HTML extended with the Angular Template Syntax defining the UI layout architecture
    - The **Style** (\*.css) file containing the cascading style sheet rules and definitions for drawing the UI
  + Here the template represents the view part of MVVM/MVC and the component represents the controller/view model
* Creating the Angular Component