**FutureValue – ASP.NET Core MVC Application:**

**Program.cs File**

This code is a C# program that creates an ASP.NET Core web application using the WebApplication.CreateBuilder method.

Graphical user interface, text, application, email

Description automatically generated

The first line creates a new instance of the WebApplicationBuilder and takes in a string array of arguments passed to the application at runtime.



Next, the code adds services to the application's dependency injection container using the Services property of the WebApplicationBuilder instance. Specifically, the code is adding controllers with views by calling the AddControllersWithViews() method. This will allow the application to handle HTTP requests and serve views to clients.

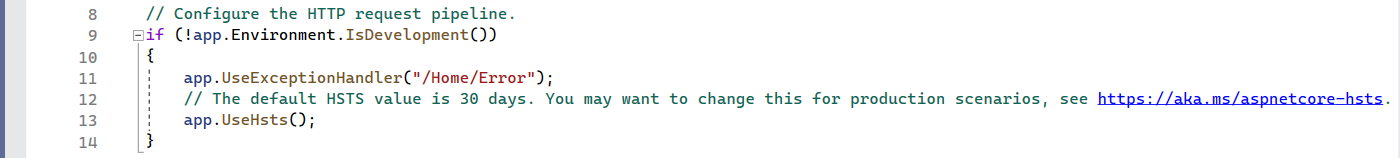
A screenshot of a computer

Description automatically generated with medium confidence

After the services are added, the code builds the application by calling the Build() method on the WebApplicationBuilder instance.



Next, the code configures the HTTP request pipeline for the application. The first step is to check whether the application is running in development mode using the Environment property of the WebApplication instance. If it's not in development mode, the application will use an exception handler to handle errors that occur while processing requests. The code does this by calling the UseExceptionHandler() method and passing in a string representing the URL of the error handling view.



The next line configures the application to use HTTP strict transport security (HSTS) by calling the UseHsts() method. HSTS is a web security policy that forces browsers to communicate with the server over HTTPS to prevent man-in-the-middle attacks.



The code then configures the application to redirect HTTP requests to HTTPS using the UseHttpsRedirection() method.



The next line configures the application to use static files by calling the UseStaticFiles() method. This allows the application to serve static content, such as images or CSS files.



The code then configures the routing middleware for the application by calling the UseRouting() method. This middleware will match incoming requests to controller actions based on the configured routes.



Next, the code configures the authorization middleware by calling the UseAuthorization() method. This middleware is responsible for authenticating and authorizing requests based on the configured policies.



After the middleware is configured, the code maps a default controller route by calling the MapControllerRoute() method. This route will map incoming requests to a default controller, action, and optional ID parameter if none are specified.

Logo

Description automatically generated with low confidence

Finally, the code runs the application by calling the Run() method on the WebApplication instance.



Overall, this code sets up an ASP.NET Core web application with middleware to handle HTTP requests, serve static content, handle errors, and enforce security policies.

**Views/appsettings.json**

This code is a configuration file in JSON format used in an ASP.NET Core application.

Let's go through it line by line:

json

Copy code

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

The above lines specify the logging settings for the application. The Logging section contains a sub-section called LogLevel, which defines the logging level for different categories of logs. In this case, the default log level for all logs is set to "Information", except for logs under the Microsoft.AspNetCore category, which are set to "Warning". This means that the application will log all messages with a severity of "Information" or higher, except for logs from the Microsoft.AspNetCore category, which will only log messages with a severity of "Warning" or higher.

"AllowedHosts": "\*"

This line specifies the allowed hosts for the application. In this case, the AllowedHosts property is set to "\*". This means that any host is allowed to access the application. If you were to deploy this application to a production environment, it would be a good idea to set this property to a specific domain or list of domains that are allowed to access the application.

Overall, this configuration file sets the logging level and allowed hosts for the ASP.NET Core application. It is important to note that this is just one of many possible configurations for an ASP.NET Core application, and that different applications may have different requirements for their logging and security settings.

**\_ViewStart.cshtml**

This code is defining the layout page that the current Razor view will use in an ASP.NET Core application.

A picture containing graphical user interface

Description automatically generated

In Razor syntax, the "@" symbol denotes a code block. The code block in this case starts with "@{" and ends with "}". Within this code block, the keyword "Layout" is used to specify the name of the layout page that the current view will use.

The value "\_Layout" is assigned to the Layout property, which is a reserved property of the Razor view that specifies the layout page for the current view. The underscore before the name of the layout page typically indicates that it is a partial view.

Therefore, this code is telling the Razor view engine to use the "\_Layout.cshtml" partial view as the layout page for the current view. The layout page will provide the overall structure and design of the HTML page, and the content of the current view will be placed within it.

**\_ViewImports.cshtml**

This code is a Razor view template file for an ASP.NET Core web application that references the FutureValue namespace and its models, and adds a tag helper from the Microsoft.AspNetCore.Mvc.TagHelpers namespace.

The @using FutureValue statement at the top of the file imports the FutureValue namespace, which presumably contains some classes and/or methods used by the view. Similarly, @using FutureValue.Models imports the Models namespace within FutureValue.

The @addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers statement adds a tag helper from the Microsoft.AspNetCore.Mvc.TagHelpers namespace to the view. Tag helpers are a way to simplify the syntax of creating HTML elements in Razor views by using custom tags that are translated into HTML by the framework at runtime.

It's not clear from this code what the specific functionality of the FutureValue application is, but this view file is likely part of a larger web application that allows users to calculate the future value of an investment based on a set of input parameters.