ASP.NET Core - True Ultimate Guide

Section 2: Getting Started - Notes

Installing Visual Studio Community and Logging In

1. **Download Visual Studio Community:**
   - Go to Visual Studio Downloads.
   - Click on the "Free Download" button for Visual Studio Community.

2. **Install Visual Studio Community:**
   - Run the downloaded installer.
   - Choose the required workloads (for ASP.NET Core development, select ".NET desktop development" and "ASP.NET and web development").
   - Click "Install" and wait for the installation to complete.

3. **Create a New Outlook.com Email:**
   - Go to Outlook Sign Up.
   - Click on "Create free account."
   - Follow the prompts to set up a new email address.

4. **Log In to Visual Studio:**
   - Open Visual Studio Community.
   - If prompted, click "Sign In."
   - Enter your new Outlook.com email and password.
   - Complete any additional sign-in steps if required.

Common Settings in Visual Studio

1. **Editor Font:**
   - Go to Tools > Options.
   - Navigate to Environment > Fonts and Colors.
   - Select Text Editor from the "Show settings for" dropdown.
   - Choose your desired font and size from the list.

2. **Theme:**
    - Go to Tools > Options.
    - Navigate to Environment > General.
    - Select your preferred theme from the "Color theme" dropdown (e.g., Dark, Light, Blue).
3. **Word Wrap:**
    - Go to Tools > Options.
    - Navigate to Text Editor > All Languages.
    - Check the box for Word wrap.

These steps should get you started with Visual Studio Community and help you configure the environment to suit your preferences.

Creating a New ASP.NET Core Empty Project

1. **Open Visual Studio Community:**
    - Launch Visual Studio.

2. **Create a New Project:**
    - Click on Create a new project.

3. **Select Project Template:**
    - In the search box, type "ASP.NET Core Empty".
    - Select ASP.NET Core Empty and click Next.

4. **Configure Your New Project:**
    - Enter the project name (e.g., "MyFirstApp").
    - Choose a location to save your project.
    - Click Next.

5. **Additional Information:**
    - Ensure .NET 7.0 (or the latest version) is selected.

- Uncheck Configure for HTTPS.
- Uncheck Enable Docker.
- Click Create.

Explanation of the Code in Detail

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();
```

1. **var builder = WebApplication.CreateBuilder(args);**
   - **Purpose**: Initializes a new instance of the WebApplicationBuilder class.
   - **Details**:
     - **WebApplication.CreateBuilder(args)**:
       - CreateBuilder is a static method that initializes a WebApplicationBuilder instance.
       - args represents command-line arguments passed to the application.
       - The builder sets up the default configuration, logging, and dependency injection (DI) services.
     - **Configuration**:
       - Reads configuration settings from various sources (e.g., appsettings.json, environment variables).
     - **Logging**:
       - Configures default logging services.
     - **Dependency Injection**:
       - Registers services to be used by the application via DI.

2. **var app = builder.Build();**
   - **Purpose**: Builds the WebApplication instance.

   - **Details**:
     - **builder.Build()**:
       - Finalizes the app's configuration and prepares it for running.
       - Compiles all middleware components added during the build process.
       - Creates the WebApplication object that will handle HTTP requests.

3. **app.MapGet("/", () => "Hello World!");**
   - **Purpose**: Sets up a route that maps HTTP GET requests to a specific path (in this case, the root URL).
   - **Details**:
     - **app.MapGet**:
       - A convenience method to define a route that matches GET requests.
       - "/" specifies the root URL path.
       - () => "Hello World!" is a lambda expression that defines the response to be returned when the route is accessed.
       - The lambda returns a plain string "Hello World!" which is sent as the HTTP response body.

4. **app.Run();**
   - **Purpose**: Runs the application.
   - **Details**:
     - **app.Run()**:
       - Starts the Kestrel web server (or the configured server) and begins listening for incoming HTTP requests.
       - This is a blocking call that keeps the application running until it is manually stopped (e.g., via Ctrl+C in the console).
       - The application is now live and will respond to requests based on the configured routes and middleware.

Summary

- The code creates and configures a minimal ASP.NET Core web application.
- WebApplication.CreateBuilder(args) sets up the application with default settings.

- builder.Build() finalizes the configuration and prepares the application.
- app.MapGet("/", () => "Hello World!") maps a GET request to the root URL and returns "Hello World!" as a response.
- app.Run() starts the web server and runs the application, ready to handle incoming requests.

Kestrel Server and Reverse Proxy Servers

Kestrel Server

Overview:

- Kestrel is the cross-platform web server for ASP.NET Core.
- It is lightweight and suitable for serving dynamic content.

Responsibilities:

- **HTTP Requests Handling**: Handles incoming HTTP requests and responses.
- **Hosting**: Hosts the ASP.NET Core application.
- **Configuration**: Supports various configurations such as HTTP/2, HTTPS, etc.

Use Case:

- Ideal for development and internal networks.
- Typically used in conjunction with a reverse proxy for production environments.

Reverse Proxy Servers

Overview:

- A reverse proxy server forwards client requests to backend servers and returns the responses to the clients.
- Common reverse proxy servers include Nginx, Apache, and IIS.

Responsibilities:

- **Load Balancing**: Distributes incoming requests across multiple servers.
- **SSL Termination**: Handles SSL/TLS encryption and decryption.
- **Caching**: Caches responses to improve performance.
- **Security**: Provides additional security features like request filtering, IP whitelisting, and rate limiting.

Use Case:

- Used in front of Kestrel to enhance security, load balancing, and other enterprise-level requirements.

Responsibilities of Kestrel and Reverse Proxy Servers

Kestrel:

- Serves HTTP requests directly.
- Provides efficient request processing.
- Should be used behind a reverse proxy for additional security and stability.

Reverse Proxy:

- Acts as an intermediary between clients and Kestrel.
- Provides SSL termination, load balancing, and security features.
- Enhances the overall performance and security of the application.

Explanation of ASP.NET Core Logs

1. Application Start Log: Listening on Port

```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5117
```

- **Category**: Microsoft.Hosting.Lifetime
- **Event ID**: 14
- **Message**: Now listening on: http://localhost:5117
- **Explanation**:
  - Indicates that the Kestrel server is now running and ready to accept HTTP requests on the specified URL and port (http://localhost:5117).
  - This log is crucial for knowing where your application is accessible.

2. Application Started Log

```
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
```

- **Category**: Microsoft.Hosting.Lifetime
- **Event ID**: 0

- **Message**: Application started. Press Ctrl+C to shut down.
- **Explanation**:
    - Confirms that the ASP.NET Core application has successfully started.
    - Provides instructions for gracefully shutting down the application by pressing Ctrl+C in the terminal or command prompt where the application is running.

3. Hosting Environment Log

```
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
```

- **Category**: Microsoft.Hosting.Lifetime
- **Event ID**: 0
- **Message**: Hosting environment: Development
- **Explanation**:
    - Specifies the current hosting environment of the application (in this case, Development).
    - The hosting environment can be Development, Staging, or Production, which affects how the application behaves, particularly in terms of logging, error handling, and configuration settings.

4. Content Root Path Log

```
info: Microsoft.Hosting.Lifetime[0]
      Content root path: c:\code\temp\MyFirstApp\MyFirstApp
```

- **Category**: Microsoft.Hosting.Lifetime
- **Event ID**: 0
- **Message**: Content root path: c:\code\temp\MyFirstApp\MyFirstApp
- **Explanation**:
    - Indicates the content root path of the application, which is the base path where the application’s content files are located.
    - This path is used to locate static files, views, and other content.
    - It helps in understanding where the application’s files are located in the file system.

Summary

- **Now listening on**: Informs you where the application is accessible.

- **Application started**: Confirms the successful start of the application and how to shut it down.
- **Hosting environment**: Indicates the environment (Development, Staging, Production) the application is running in.
- **Content root path**: Shows the base path for the application's content files.

These logs provide critical information about the state and configuration of your ASP.NET Core application, aiding in monitoring and troubleshooting.

Detailed Notes for launchSettings.json

launchSettings.json is a configuration file in ASP.NET Core projects used to define settings for how the application is launched during development. This includes settings for different environments, URLs, and other debugging options.

Structure of launchSettings.json

1. **$schema**
    - Specifies the schema URL for launchSettings.json, which helps with validation and IntelliSense support in IDEs like Visual Studio.

```
"$schema": "http://json.schemastore.org/launchsettings.json"
```

2. **iisSettings**
    - Configures settings specifically for IIS Express, a lightweight, self-contained version of IIS optimized for developers.

```
"iisSettings": {
  "windowsAuthentication": false,
  "anonymousAuthentication": true,
  "iisExpress": {
    "applicationUrl": "http://localhost:19872",
    "sslPort": 0
  }
}
```

- **windowsAuthentication**: Enables or disables Windows Authentication.
- **anonymousAuthentication**: Enables or disables Anonymous Authentication.
- **iisExpress**:
    - **applicationUrl**: The URL for the application when using IIS Express.
    - **sslPort**: The port number for HTTPS. If 0, HTTPS is disabled.

3. **profiles**
    - Defines different profiles for launching the application. Each profile can have unique settings.

```
"profiles": {
  "http": {
    "commandName": "Project",
    "dotnetRunMessages": true,
    "launchBrowser": true,
    "applicationUrl": "http://localhost:5117",
    "environmentVariables": {
      "ASPNETCORE_ENVIRONMENT": "Development"
    }
  },
  "IIS Express": {
    "commandName": "IISExpress",
    "launchBrowser": true,
    "environmentVariables": {
      "ASPNETCORE_ENVIRONMENT": "Development"
    }
  }
}
```

- **http** profile:
    - **commandName**: Specifies how the application should be launched. Project means it will use dotnet run.
    - **dotnetRunMessages**: If true, enables detailed messages from dotnet run.

        - **launchBrowser**: If true, launches the default web browser when the application starts.
        - **applicationUrl**: The URL for the application when launched directly (e.g., http://localhost:5117).
        - **environmentVariables**: Sets environment variables for the application. Here, ASPNETCORE_ENVIRONMENT is set to Development.
    - **IIS Express** profile:
        - **commandName**: IISExpress means it will launch using IIS Express.
        - **launchBrowser**: If true, launches the default web browser when the application starts.
        - **environmentVariables**: Sets environment variables, with ASPNETCORE_ENVIRONMENT set to Development.

Example launchSettings.json Code

```
jsonCopy code{
  "$schema": "http://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:19872",
      "sslPort": 0
    }
  },
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5117",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

Explanation of the Example

1. **$schema**
   - Provides IntelliSense and validation for the file.

2. **iisSettings**
   - **windowsAuthentication**: Disabled.
   - **anonymousAuthentication**: Enabled.
   - **iisExpress**:
     - **applicationUrl**: The application is accessible at http://localhost:19872.
     - **sslPort**: HTTPS is disabled (sslPort is 0).

3. **profiles**
   - **http** profile:
     - Launches using the dotnet run command.
     - Shows detailed dotnet run messages.
     - Launches the default web browser automatically.
     - Application URL is http://localhost:5117.
     - Sets ASPNETCORE_ENVIRONMENT to Development.
   - **IIS Express** profile:
     - Launches using IIS Express.

        - Launches the default web browser automatically.
        - Sets ASPNETCORE_ENVIRONMENT to Development.

Summary

- launchSettings.json configures how an ASP.NET Core application is launched during development.
- It can define multiple profiles, each with its own settings for URLs, environment variables, and launch options.
- The iisSettings section configures IIS Express settings, while the profiles section defines different launch profiles for the application.