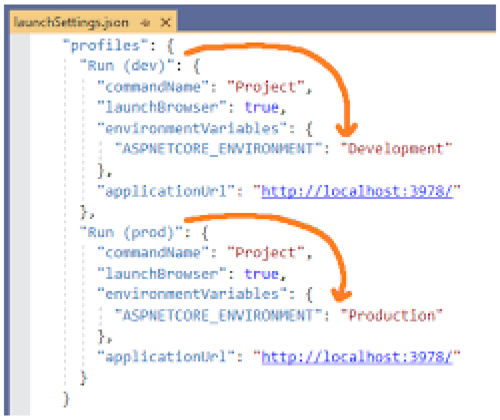
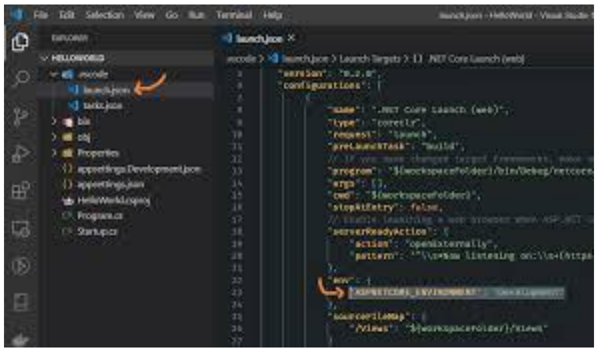
# ⭐ Configuration in ASP.NET Core – appsettings.json Explained (Instructor Style)



```
"profiles": {
  "Run (dev)": {
    "commandName": "Project",
    "launchBrowser": true,
    "environmentVariables": {
      "ASPNETCORE_ENVIRONMENT": "Development"
    },
    "applicationUrl": "http://localhost:3978/"
  },
  "Run (prod)": {
    "commandName": "Project",
    "launchBrowser": true,
    "environmentVariables": {
      "ASPNETCORE_ENVIRONMENT": "Production"
    },
    "applicationUrl": "http://localhost:3978/"
  }
}
```


File Edit Selection View Go Run Terminal Help
launch.json - HelloWorld - Visual Studio
EXPLORER
HELLOWORLD
.vscode
launch.json
tasks.json
bin
obj
Properties
appsettings.Development.json
appsettings.json
HelloWorld.csproj
Program.cs
Startup.cs
launch.json
.vscode > launch.json > Launch Targets > {} .NET Core Launch (web)

```
    "version": "0.2.0",
    "configurations": [
        {
            "name": ".NET Core Launch (web)",
            "type": "coreclr",
            "request": "launch",
            "preLaunchTask": "build",
            // If you have changed target frameworks, make
            "program": "${workspaceFolder}/bin/Debug/netcore
            "args": [],
            "cwd": "${workspaceFolder}",
            "stopAtEntry": false,
            // Enable launching a web browser when ASP.NET
            "serverReadyAction": {
                "action": "openExternally",
                "pattern": "^\\s*Now listening on:\\s+(https
            },
            "env": {
                "ASPNETCORE_ENVIRONMENT": "Development"
            },
            "sourceFileMap": {
                "/Views": "${workspaceFolder}/Views"
            }
```

ASP.NET Core uses a **flexible and powerful configuration system**. One of the most important files in this system is **appsettings.json**.

---

## ✅ What is appsettings.json?

appsettings.json is a **JSON-based configuration file** used to store:

- Application settings
- Connection strings
- Logging configuration
- Third-party service keys
- Any custom configuration your application needs

It replaces the old **Web.config** in .NET Framework.

---

## 📌 Why do we use appsettings.json?

✓ Easy to read (JSON format)
✓ Environment-based overriding (appsettings.Development.json)
✓ Supports hierarchical data
✓ Supports strong type mapping
✓ Works with dependency injection

---

## 📁 appsettings.json – Example File

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning"
    }
  },
  "ConnectionStrings": {
    "DefaultConnection": "Server=.;Database=EComDB;Trusted_Connection=True;"
  },
  "Jwt": {
    "Key": "ThisIsASecretKeyForJwtToken",
    "Issuer": "MyApp",
    "Audience": "MyUsers"
  },
  "AppSettings": {
    "ApplicationName": "E-Commerce Demo",
    "MaxItemsInCart": 10
  }
}
```

---

## ⭐ How ASP.NET Core Reads appsettings.json

In .NET 6+, the pipeline loads configuration automatically in **Program.cs**:

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


// appsettings.json is automatically loaded

// appsettings.{EnvironmentName}.json is also loaded
```

ASP.NET Core loads configuration in this order:

1. appsettings.json
2. appsettings.{Environment}.json
3. Environment variables
4. Command-line arguments

**Higher levels override lower levels.**

---

## 🌍 Environment-Specific Files

**Example:**

- appsettings.json
- appsettings.Development.json
- appsettings.Staging.json
- appsettings.Production.json

For example:

**appsettings.Development.json**

```
{
  "Logging": {
    "LogLevel": {
      "Microsoft": "Information"
    }
  }
}
```

These override ONLY the values you specify.

---

## 🎯 Reading Values in Code

### 1️⃣ Using IConfiguration (simple values)

```
public class HomeController : Controller
{
    private readonly IConfiguration _config;

    public HomeController(IConfiguration config)
    {
        _config = config;
    }

    public IActionResult Index()
    {
        var appName = _config["AppSettings:ApplicationName"];
        return Content($"App Name: {appName}");
    }
}
```

---

### 2️⃣ Strongly Typed Binding

**Step 1: Create a class**

```
public class AppSettings
```

```
{
    public string ApplicationName { get; set; }
    public int MaxItemsInCart { get; set; }
}
```

**Step 2: Register in Program.cs**

```
builder.Services.Configure<AppSettings>(builder.Configuration.GetSection("AppSettings"));
```

**Step 3: Use in Controller/Service**

```
public class CartService
{
    private readonly AppSettings _settings;

    public CartService(IOptions<AppSettings> options)
    {
        _settings = options.Value;
    }

    public int GetLimit() => _settings.MaxItemsInCart;
}
```

---

## 🔐 Connection String Example

Reading connection string:

```
var conn = builder.Configuration.GetConnectionString("DefaultConnection");
```

Used in EF Core:

```
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(conn));
```

---

### 📘 **Logging Configuration Example**

Inside appsettings.json:

```
"Logging": {
  "LogLevel": {
    "Default": "Information",
    "Microsoft.AspNetCore": "Warning"
  }
}
```

---

### 🧠 **Diagram – How appsettings.json Works**