Asp.net core Tuple Return Type

That line is using a **C# tuple return type**. It is not “two parameters”; it is **one parameter** (password) and the function **returns two values** (hash and salt) together.

static (string hashBase64, string saltBase64) HashPassword(string password)

Breakdown:

- static → the method belongs to the type (or in Program.cs, it becomes a local static method)
- (string hashBase64, string saltBase64) → **return type is a tuple** containing two strings
- HashPassword → method name
- (string password) → **one input parameter**

So it’s equivalent to “return an object with two string fields”, but lighter.

---

1) How this function is “created” (syntax explanation)

Tuple return type syntax

In C#, you can return multiple values as a tuple:

(string, string) MyMethod() { ... }

You can also **name** tuple items:

(string hashBase64, string saltBase64) MyMethod() { ... }

Those names are just labels to make the return value readable.

---

2) How to call it (and access both returned values)

Option A: Deconstruction (most common)

var (hash, salt) = HashPassword("1234");

Here:

- hash gets hashBase64
- salt gets saltBase64

Option B: Access by named properties

var result = HashPassword("1234");

Console.WriteLine(result.hashBase64);

Console.WriteLine(result.saltBase64);

Option C: Access by Item1 / Item2

var result = HashPassword("1234");

```
Console.WriteLine(result.Item1); // hash

Console.WriteLine(result.Item2); // salt
```

---

3) Why static works in Program.cs

In modern .NET templates, Program.cs uses **top-level statements**.
That means you can declare helper methods inside the same file (after your endpoint mappings).
Declaring them as static is normal and recommended.

Example layout:

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

var app = builder.Build();


app.MapGet("/", () => "OK");


app.Run();


// helper method below (valid in top-level Program.cs)

static (string hashBase64, string saltBase64) HashPassword(string password)

{

    return ("hash", "salt");

}
```

---

4) If you prefer the “traditional” way (no tuple)

Option A: Use a class

```
public class PasswordHashResult

{

    public string HashBase64 { get; set; } = "";

    public string SaltBase64 { get; set; } = "";

}


static PasswordHashResult HashPassword(string password)

{

    return new PasswordHashResult { HashBase64 = "hash", SaltBase64 = "salt" };
```

}

Option B: Use out parameters (old-style)

static void HashPassword(string password, out string hashBase64, out string saltBase64)

{

hashBase64 = "hash";

saltBase64 = "salt";

}

Call:

HashPassword("1234", out var hash, out var salt);

---

5) Why tuple return is used here

For hashing, we naturally produce **two outputs**:

- hash
- salt

Tuple return makes this clean and direct without creating an extra class.
