Migration Process Document

Title: Migration of ASP.NET MVC Controller (Framework) to ASP.NET Core Web API

Objective: Convert a traditional ASP.NET MVC controller into an ASP.NET Core Web API controller with modern conventions.

---

1. Understand the Existing Controller

Before migrating, identify:

- Controller type: Controller or ApiController
- Actions: GET, POST, PUT, DELETE
- Dependencies: Models, Services, DB Context
- Binding methods: Form data, query strings, route parameters
- Security: Authorize filters, session usage, etc.

---

2. Set Up ASP.NET Core API Project

Steps:

1. Create a new project:

```
dotnet new webapi -n ProjectName
```

2. Organize the folder structure:
   - /Controllers
   - /Models
   - /DTOs
   - /Services
   - /Data

---

3. Migrate Models and DTOs

- Copy existing models
- Replace incompatible annotations
- Introduce DTOs if needed

---

4. Convert the Controller

Example: From Framework to Core

Old ASP.NET MVC Controller:

```
public class ProductController : Controller
{
  private readonly IProductService _service = new ProductService();

  public ActionResult Index()
  {
    var products = _service.GetAll();
    return View(products);
  }

  [HttpPost]
  public ActionResult Save(Product model)
  {
    _service.Save(model);
    return RedirectToAction("Index");
  }
}
```

Migrated ASP.NET Core API Controller:

```
[ApiController]
[Route("api/[controller]")]
public class ProductController : ControllerBase
{
  private readonly IProductService _service;

  public ProductController(IProductService service)
  {
    _service = service;
  }

  [HttpGet]
  public ActionResult<IEnumerable<Product>> Get()
  {
    return Ok(_service.GetAll());
  }

  [HttpPost]
  public IActionResult Save([FromBody] Product model)
  {
    _service.Save(model);
    return CreatedAtAction(nameof(Get), new { id = model.Id }, model);
  }
}
```

---

5. Migrate Dependency Injection

In Program.cs or Startup.cs:

```
builder.Services.AddScoped<IProductService, ProductService>();
```

---

6. Handle Routing

- Use attribute routing
- No need for RouteConfig.cs
- Define [HttpGet("{id}")], [HttpPost], etc.

---

7. Replace Framework-Specific APIs

Legacy API	Replacement
HttpContext.Current	HttpContext (injected or base class)
Session	IDistributedCache or ISession
TempData/ViewBag	Strongly-typed returns
View()	Ok(), NotFound(), Created(), etc.

---

8. Test the Migrated Controller

- Use Swagger or Postman
- Test all CRUD operations
- Validate route binding and status codes

---

9. Security Considerations

- Retain [Authorize] attributes
- Configure middleware
- Support JWT/OAuth2 if applicable

---

10. Documentation and Versioning

- Use Swashbuckle (Swagger)
- Apply [ApiVersion("1.0")] if needed

Migration Checklist

Step	Status
Create ASP.NET Core API Project	Yes/No
Copy and adapt models/DTOs	Yes/No
Convert controller logic	Yes/No
Set up DI in Program.cs	Yes/No
Replace legacy APIs	Yes/No
Test endpoints	Yes/No
Implement security/auth	Yes/No
Add Swagger & versioning	Yes/No