Clean URLs And URL Mapping

لو اشتغلت على تطوير ويب قبل كده، أكيد لاحظت إن الـ URLs (عناوين الويب) بتفرق كتير في تجربة المستخدم وأداء الموقع. الـ Clean URLs (أو الروابط النظيفة) والـ URL Mapping هما أدوات أساسية في الـ web development، بتساعدك تبني روابط واضحة ومنظمة بدل الروابط المعقدة اللي مليانة رموز. يلا نفهم الموضوع مع بعض بطريقة بسيطة ومفيدة 👇

يعني إيه Clean URLs؟

الـ Clean URLs هي روابط ويب بسيطة وقابلة للقراءة، زي
/products/electronics/laptop بدل
/product.php?id=123&category=electronics&type=laptop. الروابط دي مش بس سهلة على العين، لكنها كمان بتحسن الـ SEO (تحسين محركات البحث) لأن جوجل وغيره بيفهموها أحسن، وبتقلل من الأخطاء في التنقل.

الفكرة الأساسية: بدل ما تعتمد على query strings (زي ?key=value)، بنستخدم مسارات هرمية (hierarchical paths) تعبر عن محتوى الصفحة بشكل طبيعي. ده بيحصل عادةً في الـ frameworks.

يعني إيه URL Mapping؟

الـ URL Mapping (أو رسم الروابط) هو العملية اللي بنربط بيها الروابط الخارجية (من المتصفح) بالكود الداخلي للتطبيق. يعني لما يجي طلب لـ
/user/profile/ahmed، الـ mapping بيقول للـ framework: "روح شغل الـ action ده في الـ Controller ده، ومرر الـ 'ahmed' كـ 'ID' parameter".

ده زي خريطة: الـ URL هو العنوان، والـ mapping هو الطريق اللي بيوصلك للمكان الصح. بيتم في الـ routing systems، وهو اللي بيخلي Clean URLs ممكنة عمليًا.

- **فوائد الـ Clean URLs:**

- **SEO أفضل:** الروابط الطويلة والمعقدة بتقلل من الترتيب في جوجل.

- **تجربة مستخدم أحسن:** سهل مشاركتها وتذكرها.

- **أمان أعلى:** أقل عرضة لـ injection attacks لأنها مش بتعتمد على query strings مباشرة.

- **الفرق بين Clean URLs و URL Mapping (وتكاملهم)**

Clean URLs: التركيز على شكل الرابط الخارجي (اللي يشوفه المستخدم).

URL Mapping: التركيز على الربط الداخلي (إزاي الرابط يوصل للكود).

الاثنين متكاملين: الـ Mapping بيسمح بصنع Clean URLs، ودون mapping، هتضطر تعتمد على روابط قديمة مليانة parameters. قبل ظهور الـ frameworks الحديثة ، كان الـ URLs دايمًا معقدة زي index.php?page=home&id=5، وده كان بيخلي المواقع تبدو فوضوية وصعبة الـ indexing.

الـ Clean URLs و URL Mapping في ASP.NET: إزاي بيشتغلوا؟

في ASP.NET، طريقة التعامل مع الروابط بتختلف حسب نوع الإطار (Framework) اللي بتستخدمه. هنمر على أمثلة لكل نوع:

- **ASP.NET MVC**

في MVC، الـ Routing أسهل بكتير لأن الإطار مبني على فكرة الـ Clean URLs من الأساس. الـ **Routing** بيتم في ملف RouteConfig.cs.

📌 مثال في MVC:

افترض عندك Controller اسمه ProductsController وعايز تعرض تفاصيل منتج بناءً على id أو slug.

في App_Start/RouteConfig.cs:

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "ProductDetails",
            url: "products/{id}/{slug}",
            defaults: new { controller = "Products", action =
"Details", slug = UrlParameter.Optional }
        );

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id
= UrlParameter.Optional }
        );
    }
}
```

في ProductsController.cs:

```
public class ProductsController : Controller
{
    public ActionResult Details(int id, string slug = null)
    {
        // جيب المنتج من الـ database بناءً على id
        return View();
    }
```

```
}
```

هنا، لو المستخدم دخل /products/123/macbook-pro:

- الـ id هيبقى 123.
- الـ slug هيبقى macbook-pro (اختياري، ممكن تستخدمه للـ SEO).
- الـ Controller هيستدعي الدالة Details ويعرض الصفحة.

فايدة الـ slug: تحسين الـ SEO عن طريق إضافة كلمات مفتاحية زي اسم المنتج في الرابط.

🔸 **ASP.NET Core**

في ASP.NET Core، الـ Routing بقى أقوى وأكثر مرونة، وبتدعم **Attribute Routing** و **Convention-based Routing**.

📌 **مثال باستخدام Convention-based Routing**:

في Startup.cs أو Program.cs (حسب الإصدار):

```
app.UseRouting();

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "product",
        pattern: "products/{id}/{slug?}",
        defaults: new { controller = "Products", action = "Details" }
    );
});
```

📌 **مثال باستخدام Attribute Routing** (أسهل وأكثر وضوحًا):

في ProductsController.cs:

```
[Route("products")]
public class ProductsController : ControllerBase
{
    [HttpGet("{id}/{slug?}")]
    public IActionResult Details(int id, string slug = null)
    {
        // جيب بيانات المنتج بناءً على id
        return Ok($"Product ID: {id}, Slug: {slug ?? "No slug"}");
    }
}
```

هنا، لو المستخدم دخل products/456/iphone-13/، هيظهر الناتج:

```
Product ID: 456, Slug: iphone-13
```

📌 مثال REST API في ASP.NET Core:

لو بتبني API، ممكن تستخدم Clean URLs للـ endpoints:

```
[Route("api/products")]
public class ProductsApiController : ControllerBase
{
    [HttpGet("{id}")]
    public IActionResult GetProduct(int id)
    {
        // جيب المنتج من الـ database
        return Ok(new { Id = id, Name = "Sample Product" });
    }
}
```

رابط مثل api/products/789/ هيجيب بيانات المنتج بشكل نظيف ومناسب للـ RESTful APIs.

تحسينات في .NET مقارنة بالشكل القديم

قبل ظهور الـ Routing في ASP.NET (زي في أيام ASP الكلاسيكية)، كانت الروابط معتمدة على ملفات فعلية (مثل show_product.asp?id=123)، وكان صعب تحقيق Clean URLs بدون مجهود كبير. مع تطور ASP.NET، حصلت تحسينات كبيرة:

- **استقلالية الروابط عن الملفات**:
  - في Web Forms، كانت الروابط مرتبطة بملفات .aspx. دلوقتي مع MVC و Core، الروابط بقت تعتمد على الـ Controller/Action، فبتقدر تبني هيكلية مرنة.
  - مثال: بدل /Product.aspx?id=123، بقى /products/123.
- **دعم SEO أفضل**:
  - الـ slugs (زي macbook-pro في الرابط) بتساعد محركات البحث تفهم محتوى الصفحة.
  - ASP.NET Core بيدعم تحويل الحروف لـ lowercase تلقائيًا عشان يتجنب مشاكل الـ duplicate content.
- **أداء أفضل**:
  - الـ Routing في ASP.NET Core بيستخدم خوارزميات أسرع بكتير من الـ URL Rewriting القديم في IIS.
  - الكاشينج (Caching) للـ routes بيحسن سرعة الاستجابة.
- **مرونة في الـ APIs**:
  - ASP.NET Core بيدعم RESTful routing بشكل مدمج، فبيسهل بناء APIs بحيث تكون الروابط نظيفة ومنطقية.

أمثلة عملية في ASP.NET

📌 مثال 1: متجر إلكتروني (MVC):

افترض عندك متجر وعايز تعرض منتجات حسب الفئة والمنتج.

- رابط: shop/electronics/macbook-pro/
- الـ Route:

```
routes.MapRoute(
    name: "Shop",
    url: "shop/{category}/{slug}",
    defaults: new { controller = "Shop", action = "Product" }
);
```

- الـ Controller:

```
public class ShopController : Controller
{
    public ActionResult Product(string category, string slug)
    {
        // جيب المنتج من الـ database بناءً على category و slug
        return View();
    }
}
```

📌 مثال 2: API في ASP.NET Core:

لو عندك API لإدارة المستخدمين:

- رابط: api/users/123/orders/
- الـ Route:

```
[Route("api/users")]
public class UsersApiController : ControllerBase
{
```

```
    [HttpGet("{userId}/orders")]
    public IActionResult GetOrders(int userId)
    {
        // جيب الطلبات بناءً على userId
        return Ok(new { UserId = userId, Orders = new[] { "Order1",
"Order2" } });
    }
}
```

📌 مثال 3: دعم متعدد اللغات:

لو عايز تدعم لغات مختلفة (مثل عربي وإنجليزي):

- رابط: en/products/123/ أو ar/products/123/
- الـ Route في ASP.NET Core:

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "LocalizedProducts",
        pattern: "{lang:regex(^(en|ar)$)}/products/{id}",
        defaults: new { controller = "Products", action = "Details" }
    );
});
```

هنا، الـ lang هي parameter مقيد بلغتين بس (en أو ar).

أخطاء شائعة ونصايح

🧠 الناس بتتلخبط في إيه؟

🔸 **عدم التحقق من الـ parameters**: لو id في products/{id}/ مش رقم، ممكن يسبب خطأ. استخدم constraints زي {id:int}.

🔸 **تجاهل الـ SEO**: لازم تستخدم slugs وتتجنب query strings في الروابط العامة.

⚙️ **نصايح للتعامل مع Clean URLs في .NET**:

✔️ استخدم **Attribute Routing** في ASP.NET Core لأنه أوضح وأسهل في الصيانة.

✔️ ضيف **constraints** للـ routes (زي {id:int}) عشان تحمي التطبيق من الطلبات الغلط.

✔️ استخدم أدوات زي **UrlRewrite middleware** في ASP.NET Core لو محتاج إعادة توجيه معقدة.

✔️ اختبر الروابط بأدوات زي Postman (للـ APIs) أو Google Search Console (للـ SEO).

✔️ لو بتستخدم slugs، حول النصوص لـ lowercase واستبدل المسافات بـ dashes (مثل my-product-name).