SHOWMEN TECHNOLOGIES

ASP.NET Web Forms → .NET 10 Blazor

Migration Assessment

eShopLegacyWebForms

ESTIMATED EFFORT	ESTIMATED COST	FIXED-PRICE QUOTE
78–155 hours	**$12,000–$23,500**	**$23,500**

7 pages · 2 master pages · 1 user control · 9 shared code files to update

Prepared by Showmen Technologies · hello@showmentech.com

November 13, 2026 · Confidential

Summary

This application is a small product catalog website. It lets staff list products, view details, and create, edit and delete catalog items. It has 7 pages, 2 master page layouts and 1 small reusable control, all built on the older ASP.NET Web Forms technology (.NET Framework 4.6.1). We estimate the move to .NET 10 Blazor at **78 to 155 hours ($12,000 to $23,500)**, and we offer a fixed price of **$23,500**. The biggest risks are not the pages themselves. They are the shared page layout (Site.Master), which every page depends on and which uses older techniques, and the database layer, where the way new product IDs are generated must keep working safely while the old and new sites run at the same time. We recommend going ahead with a step-by-step migration, so your site stays live the whole time and there is never a single risky "switch-over day."

Recommended approach

We will not rebuild the whole site and then switch it on all at once. Instead:

- **A reverse proxy (YARP) sits in front of both sites.** Users keep visiting the same web address. The proxy sends each request either to the existing Web Forms site or to the new Blazor site, page by page.
- **Pages move over in small groups.** At first, every request goes to the old site. As each group of pages is converted and tested, we route those addresses to the new site. If a problem appears, we can send traffic back to the old page by changing one routing rule.
- **Both sites share the same database during the transition.** This is what makes side-by-side running possible. It also means the new data layer must read and write data exactly as the old one does. This matters most for how new catalog item IDs are generated (see Key risks).
- **Read-only pages go first, then the pages that change data.** The catalog list and detail pages carry less risk than Create, Edit and Delete, so they go live first and prove the setup.
- **The application is small, so the phases are short.** Running both sites together adds a little setup work. In return, you can compare old and new pages directly and roll back immediately if something goes wrong.

Because the application has no login configured (authentication mode is not set), the proxy does not need to share sign-in cookies between the two sites. That removes one of the usual difficulties of side-by-side migration. We still need to confirm how access to the site is controlled today (see What we need from you).

Phases

Phase 1: Foundation, shared layout and proxy

- *What moves:* No user-facing pages yet. We set up the new Blazor solution, the YARP proxy and the CI build. We also rebuild the main layout and navigation from Site.Master.

- *Files:* Site.Master, App_Start/BundleConfig.cs, App_Start/RouteConfig.cs. We also clean System.Web out of the model classes (Models/CatalogBrand.cs, Models/CatalogType.cs, Models/Infrastructure/PreconfiguredData.cs) and replace the library modules: Autofac's ContainerDisposal and PropertyInjection move to built-in dependency injection, TelemetryCorrelationHttpModule and ApplicationInsightsWebTracking move to ASP.NET Core Application Insights, and Session moves to ASP.NET Core session or a replacement.
- *Depends on:* Nothing. Every later phase depends on this one.
- *Share of hours:* Covers the "New Blazor solution, layout, navigation, CI build" line (16–32 h), the "Library modules" line (2–8 h), most of the "Shared code" line (6–8 h), and Site.Master (4.5–7.5 h). This is one of the larger phases, because the layout work and the setup are done once and used by every page.

Phase 2: Data layer and read-only pages

- *What moves:* The catalog home/list page, the item details page, and the About and Contact pages.
- *Files:* Default.aspx, Catalog/Details.aspx, About.aspx, Contact.aspx. On the data side: Models/CatalogDBContext.cs, Services/CatalogService.cs, Models/Infrastructure/CatalogDBInitializer.cs and Models/CatalogItemHiLoGenerator.cs, all moving from Entity Framework 6 to EF Core.
- *Depends on:* Phase 1 (layout, dependency injection, configuration).
- *Share of hours:* The largest part of the "Data layer: Entity Framework 6 to EF Core" line (12–25 h), plus Default.aspx (4–6 h), Catalog/Details.aspx (3–5 h), About.aspx and Contact.aspx (1–1.5 h each). Default.aspx is the hardest page in this group. Its data grid and 17 inline code blocks become a Blazor table or grid component.

Phase 3: Pages that change data

- *What moves:* Creating, editing and deleting catalog items.
- *Files:* Catalog/Create.aspx, Catalog/Edit.aspx, Catalog/Delete.aspx.
- *Depends on:* Phase 2. The EF Core data layer and ID generation must be proven in production on read-only pages before any page writes data.
- *Share of hours:* Catalog/Edit.aspx (4.5–7.5 h), Catalog/Create.aspx (4–6.5 h), Catalog/Delete.aspx (3–4.5 h), plus a good part of the "Side-by-side parity testing and fixes" line (5–14 h). The Web Forms validators on Create and Edit are rebuilt as Blazor form validation. Every rule must be checked against the original behavior.

Phase 4: Mobile view, final cutover and retiring the old site

- *What moves:* The mobile layout and view switcher (or their replacement), then all remaining traffic. After that, the old Web Forms site is switched off.
- *Files:* Site.Mobile.Master (2–3 h), ViewSwitcher.ascx (1–1.5 h).
- *Depends on:* Phases 1–3, plus your decision on the mobile view.
- *Share of hours:* The "Deployment, cutover and hypercare" line (8–24 h) and the rest of parity testing. Hypercare means a period of close monitoring and quick fixes after the old site is retired.

Key risks

- **Site.Master combines several older techniques.** *Why it matters:* It is the layout every page uses. It is also the only file that uses UpdatePanel/AJAX, HttpContext.Current and Session state together. If the layout is wrong, every migrated page looks or behaves wrong. *How we handle it:* We migrate it first, in Phase 1. We find out exactly what the partial-page updates and session values do, and replace them with Blazor equivalents before any page goes live.
- **Generating new product IDs while two sites write to one database.** *Why it matters:* The app uses a custom HiLo ID generator (Models/CatalogItemHiLoGenerator.cs). It hands out blocks of IDs in advance. If the old and new sites reserve IDs differently, they could try to create two items with the same ID. *How we handle it:* The new data layer will use the same database sequence and block size as the old one. We will test concurrent creation from both sites against a copy of the database. Only then will Create go live in Phase 3.
- **The database initializer could touch real data.** *Why it matters:* Models/Infrastructure/CatalogDBInitializer.cs (301 lines, the largest shared file) creates and seeds the database. Run against production, an initializer can overwrite or reseed data. *How we handle it:* In the new app, seeding is limited to development and test environments. It will never run automatically against your live database.
- **Validation rules on Create and Edit.** *Why it matters:* These are the pages where staff enter data. If a rule is missed, bad data can get into the catalog. *How we handle it:* We list every validator in the original pages, rebuild each one, and test old and new forms side by side with the same inputs.
- **Session state is held in server memory (InProc).** *Why it matters:* The old and new sites cannot see each other's session data. If a user's journey depends on a session value set on one site and read on the other, that journey would break during the transition. *How we handle it:* In Phase 1 we find out what Site.Master stores in Session. We then group pages so that no user journey crosses between the two sites in a way that depends on session data.
- **The mobile layout may not be worth carrying forward.** *Why it matters:* Site.Mobile.Master and ViewSwitcher.ascx are an older way of serving phones. A modern responsive layout may make them unnecessary. *How we handle it:* We ask you to decide (see below). The estimate includes migrating both files, so either choice fits within it.
- **No authentication is configured in the application.** *Why it matters:* If access is controlled outside the app (for example by network rules or IIS), the new site must be protected the same way. Otherwise it could be exposed. *How we handle it:* We confirm how access works today and apply the same protection to the proxy and the new site before any traffic is routed.
- **Estimate range.** *Why it matters:* The range is wide relative to the size of the app. Most of the uncertainty is in the setup, data layer and cutover lines, not in the pages. *How we handle it:* The fixed price is set at the high estimate, so the risk of overrun sits with us, not with you.

What we need from you

- **Access:** the source code repository, the current build and deployment process, and the hosting environment, including where the YARP proxy and new site will run.
- **Database:** a recent copy of the CatalogDBContext database that we can use for testing. We also need to be told whether any other system writes to this database.
- **Monitoring:** access to the existing Application Insights resource, or permission to create a new one.
- **People:** one person who knows how catalog items are created and edited and what the validation rules mean, available for short reviews and to sign off each phase.
- **Decisions:**
  - Keep the separate mobile view, or replace it with a single responsive layout.
  - How the site's access is controlled today, and whether that should change.
  - Preferred time windows for routing each group of pages to the new site.
  - Who signs off the final retirement of the old site.

Assumptions

Included:

- All 7 pages, 2 master pages and 1 user control listed in the inventory.
- The 9 shared code files listed.
- The move from Entity Framework 6 to EF Core.
- Replacement of the listed library modules.
- Side-by-side parity testing.
- Deployment, cutover and a hypercare period.
- AI-assisted conversion, with a developer reviewing and testing every page.

The estimate assumes:

- The application works as it does today. The goal is the same behavior on modern technology, not new features.
- The database schema and data stay as they are, on the same database engine.
- The existing source code builds and matches what is running in production.

Not included:

- New features, visual redesign, or changes to business rules.
- Adding a login system. None is configured today.
- Performance or load testing beyond comparing old and new behavior.
- Hosting, licensing or cloud costs.

- Changes to other systems that use the same database.

Pricing:

- Hours are billed at $150/hour, as stated.
- The fixed-price quote of $23,500 is based on the high estimate.

Estimate

	Low	High
Pages and components	28 h	44 h
New Blazor solution, layout, navigation, CI build	16 h	32 h
Data layer: Entity Framework 6 to EF Core	12 h	25 h
Shared code: remove System.Web / HttpContext.Current dependencies	6 h	8 h
Library modules (DI, telemetry, session) to ASP.NET Core equivalents	2 h	8 h
Side-by-side parity testing and fixes	5 h	14 h
Deployment, cutover and hypercare	8 h	24 h
Total	**78 h**	**155 h**
At $150/hour	**$12,000**	**$23,500**

Fixed-price quote basis: the high estimate, $23,500. Hours assume AI-assisted conversion with a developer reviewing and testing every page.

Inventory

Kind	Count	S	M	L	XL	Code lines
Page	7	5	2	0	0	226
Master page	2	1	1	0	0	34
User control	1	1	0	0	0	37

Tiers: S under 5 points, M 5-12, L 12-25, XL 25 and up. Points come from code size, server controls and the risk factors below.

Environment

- **web.config files:** 1
- **Main web.config:** Web.config
- **Target framework:** 4.6.1
- **Authentication mode:** not set
- **Session state mode:** InProc

- **Membership provider:** no
- **Role manager:** no
- **Connection strings:** CatalogDBContext
- **Library modules/handlers:** ContainerDisposal, PropertyInjection, TelemetryCorrelationHttpModule, ApplicationInsightsWebTracking, Session
- **Custom modules/handlers:** none

Risk factors

Factor	Files
Validators	2
Inline code blocks	2
UpdatePanel/AJAX	1
HttpContext.Current	1
Session state	1
Data-bound grids	1

Hardest files

File	Kind	Tier	Hours	Why
Site.Master	Master page	M	4.5-7.5	19 code lines, 5 controls; UpdatePanel/AJAX, HttpContext.Current, Session state
Catalog/Edit.aspx	Page	M	4.5-7.5	58 code lines, 17 controls; Validators
Catalog/Create.aspx	Page	M	4-6.5	44 code lines, 15 controls; Validators
Default.aspx	Page	S	4-6	49 code lines, 5 controls; Data-bound grids, 17 inline code blocks
Catalog/Details.aspx	Page	S	3-5	21 code lines, 11 controls; 11 inline code blocks
Catalog/Delete.aspx	Page	S	3-4.5	26 code lines, 12 controls;
Site.Mobile.Master	Master page	S	2-3	15 code lines, 4 controls;
About.aspx	Page	S	1-1.5	14 code lines, 1 controls;
Contact.aspx	Page	S	1-1.5	14 code lines, 1 controls;

File	Kind	Tier	Hours	Why
ViewSwitcher.ascx	User control	S	1-1.5	37 code lines, 0 controls;

Shared code that must change

File	Lines	Hours	Depends on
Models/Infrastructure/CatalogDBInitializer.cs	301	1.5-2.5	System.Web, Entity Framework 6, ConfigurationManager
Services/CatalogService.cs	65	0.5-1	Entity Framework 6
Models/CatalogDBContext.cs	65	0.5-1	Entity Framework 6
Models/Infrastructure/PreconfiguredData.cs	49	0.5-1	System.Web
App_Start/BundleConfig.cs	43	0.5-1	System.Web
Models/CatalogBrand.cs	12	0.5-0.5	System.Web
Models/CatalogItemHiLoGenerator.cs	33	0.5-0.5	System.Web
Models/CatalogType.cs	12	0.5-0.5	System.Web
App_Start/RouteConfig.cs	40	0.5-0.5	System.Web