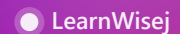


Theming & UI Modernization

Theming & the Theme Builder · Visual Hierarchy & Polishing Safely



Theming & the Theme Builder

Apply a built-in theme, see where theme files and style options fit, and use Theme Builder at a beginner level — without touching business logic.



Lesson goal

Apply a built-in Wisej.NET **theme**, understand where theme files and style options fit, and use Theme Builder at a practical beginner level — all without touching the business logic.

1. Theming vs business logic

Theming changes *how* the application looks. Business logic changes *what* it does. In a line-of-business app you should be able to improve colors, spacing, cards, headers and navigation without rewriting services, repositories, validation rules or ticket workflows.

Area	UI modernization decision	Do <i>not</i> rewrite
Theme	Choose a built-in theme such as Bootstrap-4, BootstrapDark-4, Blue-1 or Classic-2.	Ticket service, save logic, validation rules.
Spacing	Use consistent padding, margins and card spacing.	Data model classes or repository methods.
Grouping	Put related commands and information together in cards / panels.	Navigation workflow or permission checks.
Labels	Make headings, field labels and helper text clear.	The actual calculations or saved ticket records.
Commands	Place primary buttons where users expect them; keep names predictable.	Event-handler business rules.

2. Theme files, fonts, images and style options

- A theme gives the app a consistent base style for common controls.
- Fonts should be readable and consistent — avoid too many sizes or families.
- Images and icons should support the workflow, not add decoration that confuses the user.
- Style options should improve clarity: active navigation state, clear section headings, readable buttons.

- Keep one visual system — same button sizes, card padding, label pattern and color purpose across screens.

3. The Theme Builder concept

Theme Builder is for when the built-in themes are close but not exactly what the project needs. A beginner should first apply a built-in theme, then make small practical adjustments — primary color, backgrounds, font sizing, control spacing. Don't start by hand-editing every screen.



Start with a built-in theme

Pick a built-in theme first and adjust a few values. Reach for Theme Builder only when the base theme can't get you the rest of the way.

4. A modern screen structure

A modern business screen usually has a clear header, left navigation, a main content area, grouped cards and predictable command placement, so users always know where they are and what to do next.

```
Header      → title, module label, current theme selector
Navigation  → Dashboard · Tickets · Customers · Reports · Settings
Content area → page heading + short description
              → a row of metric cards (large numbers)
              → grouped commands | recent activity
```



You're ready for the next reading

You can separate theming from business logic, apply a built-in theme, and describe a modern screen structure. Next: the visual-hierarchy rules and how to polish a screen without breaking what works.

Visual Hierarchy & Polishing Safely

Visual-hierarchy rules that make screens easy to scan, plus how to polish the layout without breaking binding, navigation or validation.



Lesson goal

Use visual-hierarchy rules to make screens easy to scan, then modernize the layout **without** breaking data binding, navigation, validation or business rules.

1. Visual hierarchy rules

Rule	What it means	Example in Module 7
Make the page title obvious	The user should immediately know which screen they're on.	Dashboard, Tickets, Customers, Reports, Settings headings.
Group related content	Use panels / cards so related records and commands stay together.	Metric cards, ticket-queue card, UI settings card.
Keep commands predictable	Put buttons near the content they affect.	Create / Edit / Close Ticket under the ticket queue.
Use consistent spacing	Screens should feel like the same application.	Same left padding, card widths and row spacing across pages.
Show active navigation state	The selected navigation button should look selected.	Green selected button with a border on the current page.

2. Polish without rewriting logic

- Rename controls clearly if needed, but avoid breaking existing event handlers.
- Move related controls into panels or cards without changing the service layer.
- Use `Dock`, `Anchor` and `AutoSize` intentionally so the screen stays usable when resized.
- Keep the same data source and binding logic when modernizing grids, lists and detail panels.
- Test each navigation button and workflow after the UI is polished.



Beginner mistake to avoid

Don't hide broken logic behind a nicer theme. A modern UI still needs working navigation, validation, save/cancel behavior, safe errors and clear status messages.

3. Keep theme settings out of business logic

A theme selector is a UI setting — keep it separate from the ticket service. Apply the theme and update a small status label; don't entangle it with save or validation code.

```
private void cboTheme_SelectedIndexChanged(object sender, EventArgs e)
{
    string selectedTheme = cboTheme.Text;

    // Apply the selected theme – keep this UI setting separate
    // from ticket service logic.
    Application.Theme.Name = selectedTheme;
    lblCurrentTheme.Text = "Current theme: " + selectedTheme;
}
```

4. Module 7 checkpoint

- Can you explain what the theme changes?
- Can you point to the business logic that should *not* change?
- Can you identify the current page from the navigation state?
- Can you explain why cards, labels and spacing make the app easier to use?



You're ready for the lab

Next: watch the modernization walkthrough, pass the knowledge check, then theme and modernize the dashboard — apply a built-in theme, group metrics into cards, show the active nav state, and keep every existing workflow working.